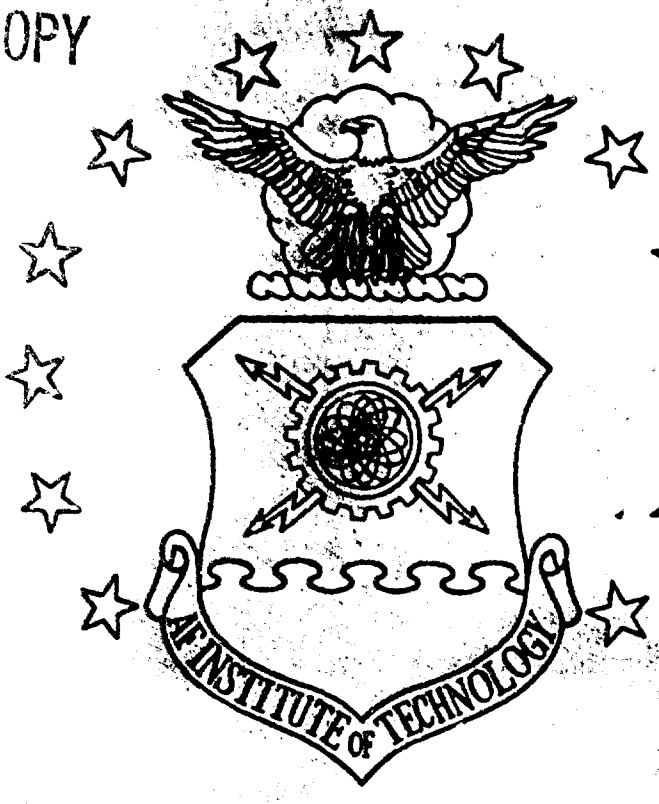
DTIC FILE COPY

AD-A231 030

(1)



DTIC
ELECTE
JAN 22 1991
S B D

AN OBJECT-ORIENTED MILITARY SIMULATION BASELINE FOR PARALLEL SIMULATION RESEARCH

THESIS

Robert J Rizza
Captain, USAF

AFIT/GCS/ENG/90D-12

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
Approved for public release;

Best Available Copy

AFIT/GCS/ENG/90D-12

AN OBJECT-ORIENTED MILITARY SIMULATION BASELINE FOR PARALLEL SIMULATION RESEARCH

THESIS

Robert J Rizza
Captain, USAF

AFIT/GCS/ENG/90D-12

DTIC
ELECTE
JAN 22 1991
S B D

Approved for public release; distibution unlimited

AFIT/GCS/ENG/90D-12

AN OBJECT-ORIENTED MILITARY SIMULATION BASELINE FOR PARALLEL SIMULATION RESEARCH

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Systems)

Robert J Rizza, BS
Captain, USAF

December, 1990

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Approved for public release; distibution unlimited

QUALITY INSPECTED 4

Acknowledgments

I would like to thank my advisor, Dr. Thomas Hartrum, for his input during this thesis effort. I would also like to thank my committee members, Major Kim Kanzaki and Captain Catherine Lamanna for their inputs.

I would like to thank the entire GCS section for all their help and support, but I would especially like to thank Captain Ann Lee. Without Ann I sincerly believe that I may not have made it through to graduation. Her support, encouragement, and faith, gave me the strength to endure. Her ever present smile and laughter gave me something to look forward to and something to count on at those times when things looked their bleakest. I could say much more but instead I'll just say this: I will always feel close to Ann, and never forget the friendship we shared.

I want to thank the graphics guys, those C gurus, who so many times bailed me out when all my efforts at debugging failed. Without the help of Captains Ed Williams, Dave Dahn, and Phil Platt, I could very easily still be debugging today.

I want to thank my neighbor, my friend, and my fellow corvetter, Skip Layfield for his support and encouragement. Those times spent working together on my 70 always brought me back to earth.

My four year old son Keith also deserves thanks. He was totally oblivious to what was going on and that is what I thank him for most. Innocence is beautiful.

I don't know how to thank my wife. She may of had the most difficult job of any of us. Having to stand by and watch me suffer, unable to help, was extremely difficult for her. She was always there ready to give her love, hoping that it would in some way lessen my load. Although it may not have showed, it did more than help, in fact without her presence I would have been lost. Thank you Kathy, I love you.

Robert J Rizza

Table of Contents

List of Figures

List of Tables

Abstract

This ~~paper~~ thesis documents the design and implementation of a discrete event military simulation using a modular object-oriented design and the C programming language. The basic simulation is one of interacting objects. The objects move along a predetermined path until they encounter another object. Objects react to the encountered object according to the implemented algorithm. Object reaction options are fight, evade, or do nothing. In the code's current form it is generic enough to allow a user the flexibility of creating an infinite number of scenarios bounded in size by the hardware's memory capacity. The modularity of design will allow for easy expansion of object complexity and detail, as well as easy removal or replacement of functions or events. The simulation code makes use of a generic linked list data structure and simulation driver. This adds yet another area to the code where expansion, removal, or replacement could be easily accomplished. The net result is a military scenario simulation program which is highly expandable and modifiable, yet compact enough to be easily understood.

AN OBJECT-ORIENTED MILITARY SIMULATION BASELINE FOR PARALLEL SIMULATION RESEARCH

I. INTRODUCTION

This thesis deals with one part of the ongoing research effort investigating possible run-time speedup of military simulation software using parallel processing. Currently, a shortage of military simulation software for use in Air Force Institute of Technology (AFIT) research exists. The purpose of this thesis is to provide a new source of this software.

1.1 Background

Recent development of high speed parallel and distributed computer architectures have spawned a new interest in the simulation world. Fujimoto believes that these new architectural designs can dramatically speed up the run-time of many computationally intensive problems such as those in large simulations (12:19). The benefits of speedup are twofold. First, speedup would enable existing simulations to run at higher speeds, allowing for quicker decision making or enough time to make additional simulation runs. Second, speedup would allow for the development of more complex, and ideally, more accurate simulations.

At present, the Air Force Institute of Technology (AFIT) does not have the ability to explore the applicability of parallel or distributed simulations dealing with a military scenario.

In general there are three requirements needed before one can study parallel or distributed computer simulations.

- First, the study requires a computer with a parallel or distributed architecture. AFIT has four such systems. Two are Intel Hypercube iPSC/1s, each using thirty-two INTEL 80286 microprocessors, one per node. One iPSC/1 has a vector processor at each node and the other has an expanded memory capacity at each node. AFIT also has an Intel Hypercube iPSC/2 which uses eight of Intel's 80386 microprocessors, one per node. Lastly, AFIT has an Encore Multimax which uses a shared memory architecture with sixteen nodes.
- Second, a software package to handle the protocol used for node intercommunication (message passing between the nodes (processors)) is needed. AFIT is currently using a software package called Spectrum, a parallel simulation testbed developed by Paul Reynolds at the University of Virginia (UVA) (24), which is written in the C programming language.
- Lastly, the study requires a simulation which is computationally intensive, has a significant amount of code which need not be run sequentially, and is compatible with the software used to handle node intercommunication (in this case, compatible with Spectrum).

Parallelizing a simulation can be studied using many types of simulations. However, the area of particular interest to AFIT is parallelizing battle and other military scenario simulations. AFIT's interest in this area of study stems not only from the fact that the typical AFIT student is in the military, but from specific interest and requirements from sponsoring organizations, as well as the opportunity to explore claims made by Nicols as to the limitations imposed on parallelizing an event driven battlefield simulation (21:141). AFIT does **not** have a military simulation which is appropriate for parallelization. Because of the lack of a military simulation AFIT has only a limited ability to explore the applicability of parallel or distributed architectures to simulation software. In addition to the requirements stated above, the simulation must also adhere to the following:

- It must contain the types and number of constructs which according to current literature pose a problem to parallelization.
- It should produce as an output the information needed to display the simulation.
- The code shall be easily modified, maintained, and reused, since it will be restructured in various parallel configurations.

To meet AFITs needs three options exist:

1. Find an existing simulation which meets the stated requirements.
2. Modify an existing simulation to meet the stated requirements.
3. Create a new simulation which meets the stated requirements.

In regard to the first two items, there are a number of military simulations currently in use in the field, but the following constraints preclude them from use.

- Most current military simulations are coded in Fortran which is not compatible with UVA's simulation testbed Spectrum. Spectrum is coded entirely in C.
- Most current military simulations are very large, and have been built over time by different programmers. This type of construction makes translation to C and parallelization nearly impossible.
- Many of the current military simulations are classified, making it difficult if not impossible to use them in AFIT's parallel processing laboratory.

Thus the only solution is option three. The rationale of this thesis is therefore straightforward: without a military scenario simulation which meets the basic requirements stated earlier, no further research into run-time simulation speedup can be made.

1.2 Problem Statement

Design and implement a discrete event military scenario simulation using a modular object-oriented design approach and the C programming language.

1.3 Research Questions

Answers to the following questions are part of this research effort:

1. Can a discrete event military scenario simulation be written in C using a modular object-oriented design approach?
2. What types of issues and constructs are currently viewed as possible problem areas to the parallelization process?
3. What information needs to be provided to a remote graphical interface system?
4. What, if any, real-time simulation inputs should the user be allowed to make?

1.4 Definitions

Discrete Event Simulation A simulation in which dependent variables change discretely at specified *points* in simulated time called event times (23:62) (20:135).

Event Something which causes change in the state of an object or entity (20:136).

Object An entity which has a state and a defined set of operations to access and modify that state (28:204) (6:20).

Object-Oriented Design A design approach where the system is viewed as being composed of interacting objects instead of a group of interacting functions (25:277).

Time Driven (continuous) Simulation A simulation where dependent variables may change continuously over simulated time at set time increments (23:62).

1.5 Assumptions

Several assumptions were made concerning this effort. First, the simulation developed here will be used solely for research purposes with the intent of establishing *feasibility* of operational applications of parallel or distributed architectures to simulations, and in particular, military simulations. This assumption directly affects issues of of scope. Second, DeRouchey's work in developing a generic graphical display driver (9) will support this simulation. Lastly, Spectrum (24), or a comparable software package written in C, will be available for use during the follow-on research which uses this code.

1.6 Scope

The extent of this work is restricted by the following limitations:

- This simulation is written to run on a single serial processor. Parallel issues will be considered during all phases of design and development, but this code will not be parallelized as part of this thesis.
- The simulation should be a "representative" military simulation, but time fidelity and object characteristics are not goals.
- The simulation should provide output in a fashion that can be used by a generic graphics display, but not be constrained by this interface. Because the graphics driver is a separate but concurrent effort by DeRouchey (9), this work should be independent of the graphics work.

II. LITERATURE REVIEW

2.1 Introduction

The purpose of this chapter is to at least summarize some of the current literature concerned with simulations in general, event driven simulations, and object-oriented simulations. Since C is the required implementation language as explained in Chapter One, its applicability to object-oriented programming will also be explored.

All of the above topics have already been thoroughly addressed and are well understood. It is not the purpose of this chapter to imply that work of this type has never been done before. However, as described in Chapter One, there exists a specific requirement for a simulation which is:

- a military scenario
- event driven
- written in C
- highly modifiable and expandable
- is or could create a high computational load
- compact enough to be understood by one person

A simulation fitting this bill was not found, thus creating the need for this new work.

2.2 Background

According to Thesen and Travis, simulation in its broadest sense is a performance analysis tool which is used as a decision aid (29:7). Almost any question can be answered by a properly designed simulation. The proper design of a simulation

can only be done if the problem is completely understood. Thesen and Travis go on to point out some common pitfalls to any simulation. First, creating a simulation is an art, requiring a special talent. The quality of any analysis depends on the quality of the model. Second, sometimes it is difficult to determine if a particular observation made during a simulation run is representative of the system behavior because of the use of randomness in the simulation (29:7-8). To help avoid the first pitfall, the programmer must use care, patience, and attention to detail while in the creation phase. The second pitfall is one of interpretation, not coding, and should be easily handled if the programmer does not forget what degree of randomness has been implemented in the simulation under study.

As defined in Chapter One, and as described by many other authors, a discrete event simulation is a simulation where time is updated as events occur and not at some predetermined time step (23:62) (20:135) (17:11). In this scheme, events are processed as quickly as possible, effectively deleting the "dead time" between events. Events can occur at irregular time intervals which are at least, in part, determined by what are defined as events. Consider the following example of a discrete event simulation: A tank moves in a straight line for 100 miles. If the only event defined is "reached_turnpoint" then this simulation has zero events and the simulation time clock is never updated. However, if "travelled_one_mile" is an event, then this simulation will have one hundred events, and the time clock should reflect the time of the last event.

There has been enormous amounts of information published in recent years on the topic of object-oriented design. Object oriented design is one in which the design focuses on objects rather than functions, with messages passed from object to object (25:277). Objects are entities which have a state, a defined set of operations to access or modify it, are denoted by a name and have restricted visibility of and by other objects (28:204) (6:20) (7:48-50). Booch is probably one of the most widely recognized proponents of object-oriented design today. His books *SOFTWARE COMPO-*

NENTS WITH ADA and *SOFTWARE ENGINEERING WITH ADA* do a good job describing why, and how, to use an object-oriented approach (6) (7). Now that a common baseline has been established, the discussion can turn to some of the current work relating to this effort.

2.3 Related Work

2.3.1 **Simulations in General** Simulations are much older than the oldest mechanical computer. Indeed, man has probably been simulating from the point at which he gained the ability to think abstractly. Anytime a person thinks ahead to "imagine" the consequences of an action, or sequence of actions, that person has basically run a simulation, using their brain as the information processor. Today, with the help of computers, we are able to simulate actions, or a sequence of actions, which for reasons of complexity, may not be able to be simulated in a single person's head.

Simulations, in general, are so well understood that they will not be detailed here. If more information at this level is needed, the references of Pritsker and Neelankavil should suffice. However, the article by Thesen and Travis presented some valuable information about not only what simulations were, and what they are used for, but what to keep in mind as one develops a simulation. Those suggestions were (29:13):

- Define your objectives before simulating.
- Use the correct level of detail – begin with a simple model.
- Select software that is appropriate to your problem, level of experience and time frame.
- Remember that simulation results may be observations of random variables, and interpret your results accordingly.

The following subsections present highlights and pertinent information from articles, papers, or books in the areas of object-oriented simulation, military modelling, C as it pertains to discrete event simulations and object-oriented design, and some background on parallel simulations.

2.3.2 **Object-Oriented Simulation** The following papers address object-oriented simulations.

A Perspective on Object-Oriented Simulation (25)

Probably the most important point made in this paper, as it pertains to the work of this thesis, is that an object-oriented design fits well into how most things to be simulated are viewed. To be more specific, one can very naturally view something to be simulated as a group of objects, or things, that do something or may have something done to them. Thus, they have legal operations which they can do (e.g. the object aircraft might be able to turn, fire a missile, or land), or can be done to them (e.g. the same aircraft may be fired on by another aircraft). Objects also have a corresponding state before, during, and after the operation. Roberts and Heim go on to point out construction of objects in this manner help to modularize the problem in its earliest stages of analysis. A second major advantage to object-oriented design is that simulations become more easily extensible. This is, of course, a desired feature of the simulation written as part of this thesis work. A last important advantage pointed out in this paper is that objects provide a natural baseline for concurrency. The idea here is that each object, or subset of objects, could be assigned a particular processor of its own and work away until communication was needed.

Design and Implementation Issues in Object-Oriented Simulation (5)

Bezevin points out an important aspect of coding a simulation. First and foremost is the principle of readability. Having readable code is always important and is obvious to anyone who has given a copy of their code to someone else to use, but it's is of paramount importance to simulations and the work of this thesis

in particular. In general, the only way to determine if a simulation is modelling something correctly is to go back and look at the code. Unverifiable simulation code is not worth much if real decisions are to be made based on its output. Bezivin's point is well taken here because not only is a simulation going to be produced as part of this thesis effort , but it is known that the code will be used in follow-on work and thus must be readable. The second point Bezevin makes is that simulation code should be efficient. While this is certainly true, especially for large simulations where time may be a critical factor, it is luckily not a requirement of the simulation developed for this thesis. On the contrary, a high computational load is desired since eventually this code is to be used to study speedup by parallelization. The second half of Bezivin's paper deals with how the object-oriented parallelization can help meet the needs of the sometimes opposing objectives of readability and efficiency. As found in many of the other references, the main thrust is that by providing a good object model (e.g. objects, and operations), the code naturally is easier to follow and normally more efficient.

Some Experiments in Object-Oriented Simulation (4)

In this paper, Bezivin actually focuses on revealing greater flexibility of the Smalltalk-80 [1] simulation language. Languages specifically designed for use in creating simulations are plentiful (23) (26) (14) (22); however, they are not the focus of this research. Although Bezivin's paper revolved around Smalltalk-80, it did give valuable insight into a type of simulation where there are basically two types of entities, clients and servers. Clients are active entities and the servers are passive. In the example given, Bezivin modelled vehicles travelling between cross road junctions as active entities, and the junctions themselves where passive entities. This type of concept may be applicable to this thesis work depending on final design decisions. Bezivin also spent a good deal of time on semaphores and monitors, but at this time there are no plans to use shared data, so there was limited applicability of this

[1] Smalltalk-80 is a trademark of the Xerox Corporation

data. However, the Air Force Institute of Technology does have a shared memory parallel architecture computer, and if it is used in the follow-on work to this thesis, the information from this paper will be of value.

2.3.3 **Military Simulation Modelling** The intent of this section is to present some of the ideas or concepts of how military models are constructed or appear, either through a specific example or background data. A military simulation is "a type of model in which the objective is generally to replicate a reasonably well understood process, and for which uncertainties are treated by Monte Carlo method." (2:14).

The TAC Brawler Air Combat Simulatio (3)

TAC Brawler is a simulation of air-to-air combat capable of handling 2 - 32 aircraft. It is written in Fortran and has over 150,000 lines of code. In almost all cases, the characteristics and behavior and reactions of TAC Brawler entities are much more detailed than the planned objects in the simulation to be developed. However, it is interesting to see how TAC Brawler models certain characteristics. For example, missiles and guns are both modelled. Missiles take into account guidance, seeker, envelope and fuzing. Sensors modelled are eyes, radar and Infra Red Search and Track (IRST). Communications are explicitly modelled as well as Identify-Friend or Foe (IFF), defensive avionics, radar jamming and Missile Approach Warning (MAW). This paper has a wealth of information on what things can be simulated as well as limited information on how it is done.

Two Aggressive Aircraft in a Realistic Short-Range Combat as a Differential Game Study (16)

In this paper, Jarmack offers a rigorous mathematical solution to a close aerial combat with IR missiles. The level of detail, not to mention its complexity, of the material presented is beyond that which is planned for this thesis work. However, if

at a later date, a more rigorous solution is needed in this area, this approach may be applicable.

Simulation of Multiple Aircraft Information, Communication and Decision in Air Combat (8)

This was an extremely good article on the modelling of communication and decision making process. Although it is not planned at this time to model communications in the simulation to be developed for this thesis, this is one area which may be considered for consideration if time permits. Chan and Vogel also give a good example of a decision tree which establishes how to assign target priorities. Undoubtedly, a tree of this nature will be used to determine targets in the simulation being developed.

Military Make-Believe (1)

This was really a survey article of what the state of art simulators had to offer. This was a good background piece, but did not give much specific insight to simulations at the level of the work of this thesis. The article's focus was on big system simulators such as interactive simulators for the Mirage F1 fighter, AH-64A Apache helicopter, or the Mirage 2000 fighter.

2.3.4 **Some insights to C** C is a general-purpose programming language which is touted for its portability, flexibility and power (30:4). It features economy of expression, modern control flow, data structures, and has a rich set of operators (18:xi). "C was originally designed for and implemented on the UNIX operating system on the DEC PDP-11, by Dennis Ritchie The operating system, the C compiler, and essentially all UNIX applications programs are written in C"(18:xi). C is widely used, and has gained even more popularity as versions became available for use on personal computers. The following articles or papers deal with the application of C to object-oriented programming or discrete event simulations.

It's an Attitude (19)

In this article, Linowes sets out to describe one way in which C can be used to do object-oriented programming. As is planned for this thesis work, Linowes uses C structures as templates of objects. An instantiated structure thus creates an object. Linowes also formalizes a message passing scheme for communication between objects. While this makes clear the communication between objects, it is felt at this time that it may add an unnecessary level to object interaction. Instead of sending a "message" containing what operation is to be performed, it may be simpler to just make the operation to be performed the message. Linowes also illustrates how he handles inheritance of attributes to subclasses of objects. Basically, he uses a strategy of #include chaining, where in the structure definition of one object he uses #include to include another file thus enabling inheritance to occur. This seems like a reasonable approach to inheritance if it is used in this thesis work.

Object-Oriented Programming As a Programming Style (32)

White's article was another example of how C could be used to code using an object-oriented approach. White is a little more detailed in his coverage than Linowes, but is nearly the same in how he handles messages and inheritance. White does separate "messages" from what he calls "methods" where Linowes does not. In White's version, "messages" get sent to an object, where something decides what "method" (operation) to invoke. It really just seems to be a rather minor difference, but it is a slightly different twist. The rest of White's article focused on C++ and how it can be used in object-oriented programming.

C Based Discrete Event Simulation Support System (27)

This paper describes a C based simulation environment for creating and executing discrete-event simulation models in which the event routines are coded in C. The system described by Selvaraj et.al. is divided into eight task modules. The two most important modules are the executive controller and the memory management module. The executive controller module is similar to the Generic Simulation Driver in the appendix of this thesis. It basically executes the simulation, placing and taking event entities off the "simulation calendar" (event list) until no more events exist to be executed or the simulation gets a termination event. In the Generic Simulation Driver memory management is not handled as a separate issue; instead it is done on an as-needed basis within the code.

2.3.5 **Parallel Simulation** While writing a parallel simulation is not the objective of this thesis work, writing a simulation that can be parallelized certainly is. Thus, some knowledge of what parallelization may entail is definitely an area to be considered.

A Survey of Parallel Computer Architectures (10)

Duncan starts his article off by addressing Flynn's taxonomy (11). Flynn classifies architectures on the presence of single or multiple data streams of instructions and data. Flynn's MIMD (multiple instruction, multiple data stream) machines are the types of parallel computers available at the Air Force Institute of Technology, and as such, will be what is discussed here. MIMD machines involve multiple processors autonomously executing diverse instructions on diverse data. MIMD architectures are generally more complex then machines of Flynn's other classifications, but MIMD machines can also mimic the other machines behavior if necessary. The next important area within MIMD machines deals with whether the machine has a shared memory approach, where all the processors have immediate and direct access to some central memory, or whether the machine has a distributed memory scheme.

whereby each processor has its own memory and access to another processor's memory is indirectly through some type of message. As stated in Chapter One, the Air Force Institute of Technology has both distributed memory machines and a shared memory machine. It is apparent at this point that a significant problem to be avoided when designing any program which will be run on a distributed memory machine is the use of global variables. It should be obvious, but excessive use of globals will create a large communication overhead caused by message passing to keep variables updated. Shared memory machines do avoid this shortfall, but in shared memory machines other problems like data access sychronization must be solved.

Parallel Discrete Event Simulation (12)

This paper deals with the execution of a single simulation program application in a set of concurrently executing processes, or more simply put, the parallel execution of a single simulation. More interesting than the general objective of this paper was the section called, "Why Is Parallel Discrete Event Simulations Hard?". Fujimoto points a finger immediately to global data structures, but of course this is not surprising. He also discusses how hard it is to ensure the proper execution sequence of events, pointing out that the constraints that dictate which computations must be executed before which others is often quite complex and data dependent. It is here that overlap into the synchronization area becomes evident. It appears, however, that clever partitioning of the problem may help to alleviate part of the precedence problems.

An Empirical Study of Data Partitioning and Replication in Parallel Simulation (31)

Wieland's article looked at the issue of partitioning in a parallel simulation. In particular he focused on the issue of proximity detection between objects in adjacent sectors, where sectoring has been chosen as the parallel partitioning strategy. An obvious strategy to handle movement between sectors is to create an event corresponding to travel across a boundary. At that event time an object could be "handed over" from the processor currently controlling the object to the processor controlling

the new sector. This strategy is straightforward enough, but the more subtle issue is how to handle detection between objects that are near the borders but still in different sectors. Wieland mentions a number of strategies like use of a buffer zone between sectors, overlapping sectors, or data replication as an object and its sensor zone move from one sector to another. This last strategy intrinsically sounds best since sensor zones can then be of differing sizes as is not the case for the other strategies. One last note of particular interest are his comments on proximity detection within a sector. Wieland comments that a quadratic equation can be used to solve for the time at which two objects will first come in contact with one another and the time at which they will lose contact with each other. This notion will be further explored as the design of the simulation for this thesis progresses.

III. THE MODEL

3.1 Introduction

This chapter is broken into three distinct areas: the overall model of battle or high level design, a more detailed look at the model of battle or low level design, and an explanation of program interfaces. In the high level design area there are three topics of discussion, the objects which will be available for instantiation and use in a given scenario, the events which may affect object attributes, and the basic models for how objects perceive, move, and fight. In the low level design area objects will be viewed in detail, and the functions which support the events will be discussed. The object discussion will include attributes, and rationale for the object's existence. The final area defines the interface between the simulation driver (what executes the simulation), and the actual simulation.

3.2 System Overview

Here is the "big picture", without regard to describing how the code is actually accomplishing any of these actions. Figure 3.1 illustrates the big picture as seen from the outside. The user must create a scenario file (as described in this chapter and Appendix F). Once created, the executable rizsim simulation code is executed using the created scenario file. The simulation produces an output (at this time a file), which is read by the display driver which graphically displays the simulation output.

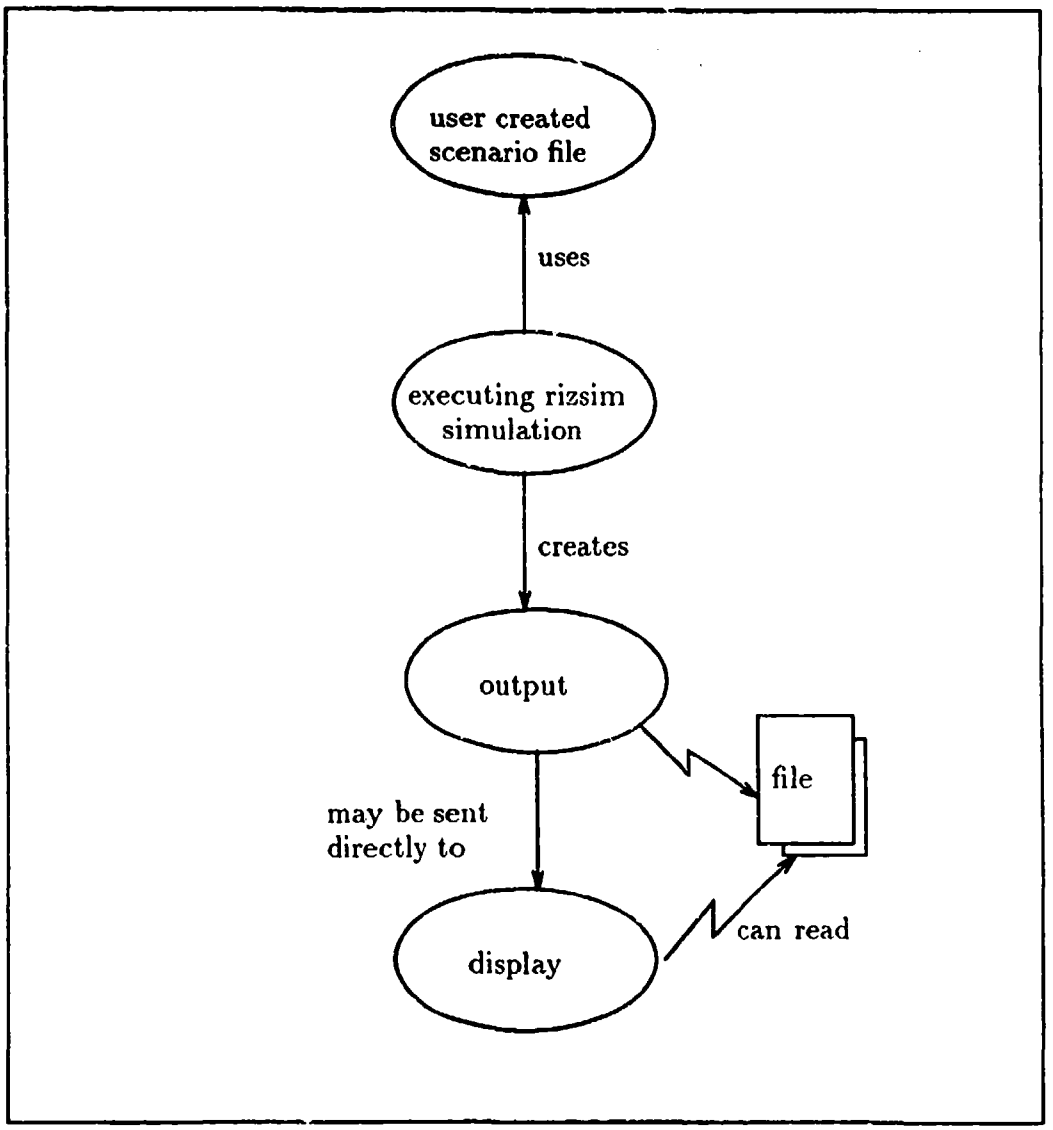


Figure 3.1. The "Big Picture"

Figure 3.2 depicts a simple two dimensional representation of a typical simulation scenario. At time t there are eight objects in the simulation, a flight of three aircraft approaching from the southwest, a single ship approaching from the northwest on an intersecting path with the flight of three, three tanks moving in a northwestly direction, and one other single ship moving northwest. At time Δt later two of the flight of three have been destroyed as well as the single ship attacker. Now only one of the flight of three remains, along with the three tanks and the other single ship. By some other Δt later, the remaining single ship has turned north to evade the other aircraft as the other aircraft flew by. On the last leg of the single ship's journey, the single ship destroys the three tanks.

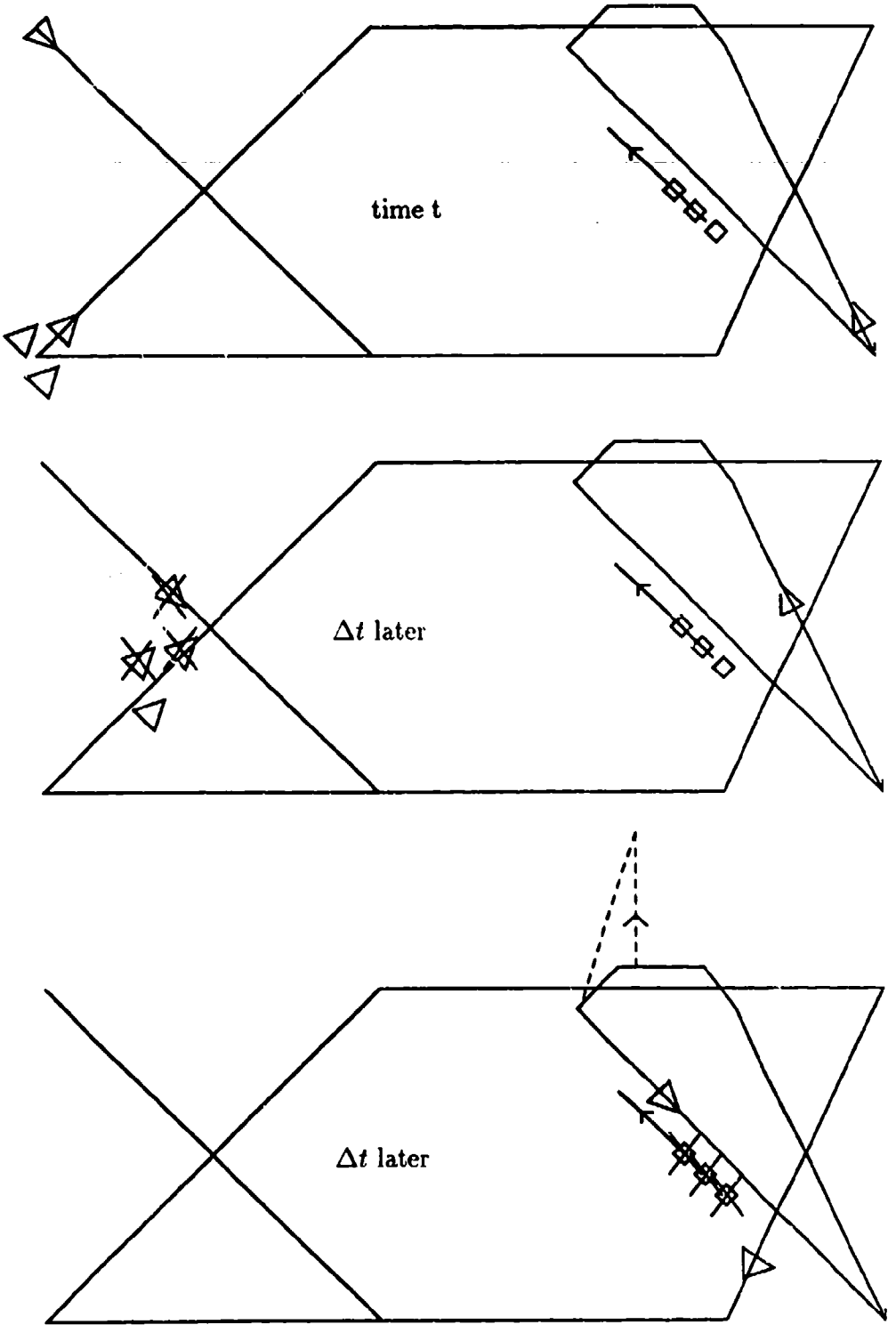


Figure 3.2. Depiction of a Typical Scenario

3.3 High Level Design

The general system model is one of interacting objects. Moveable objects (vehicles) have a predetermined route as part of the input scenario, which may or may not be altered depending on obstacles or threats encountered. Moveable objects may have a predetermined target or destination as an objective, or may be in search of a target of opportunity. Stationary objects, such as Surface-to-Air Missle (SAM) sites, will attack any valid target within range if the site has the resources to do so. Once the simulation has begun, objects move along their predetermined routes carrying out their respective missions. If an obstacle or threat is encountered along the route, an event (e.g. entered_sensor_range) is scheduled to handle the situation. The vehicle will choose to either attack, evade, or take no action, in response to the obstacle or threat. Although all objects are autonomous entities reacting to threats or obstacles separately, it is expected that similar objects (e.g. two F-15s flying the same route with a half second seperation) will react similarly to the same threat or obstacle. This is because the algorithm used by the F-15s to determine their course of action will be the same. The simulation will continue until a termination event is executed or no more events are pending in the Next Event Queue (NEQ). A termination event can be scheduled at any time by using the end_sim function available in the generic simulation driver.

3.3.1 **The Objects** The design strategy used in defining objects for this simulation was to keep the objects as generic as possible without being unrealistic as to the breadth of application of any one object. Stated simply, there is no "super" object that can be instantiated to create any entity type in the system. However, many of the semi-generic objects will be able to be instantiated to create a limited number of seemingly different object types. A good example of this is the object "sensors" which can be instantiated as a number of different sensor types from eyes to radar.

The objects are those entities within the simulation scenario which make up the "order of battle". The order of battle as used here refers to the types and amounts of instantiated objects which will be players in the scenario to be run. Table 3.1 lists all the object types which are available to the user for instantiation. Figure 3.3 shows the relationships between the objects in a given scenario. Object-attribute relationships are addressed in the low level design section.

Object Types
object_attributes
performance_characteristics
sensors
armaments
defensive_systems
route_data
operator
target_list
master_obj_list

Table 3.1. **Simulation Object Types**

Objects instantiated using the "object_attributes" type are probably the most important of all the objects within the scenario. It is the movement of these objects which gives the simulation much of the computational complexity sought by this work. The object_attributes type, as are many of the other object types, is a skeleton definition where the user fills in the applicable attributes with the correct values when the object is instantiated. By simply providing zeros as the velocity vector attributes and no route points other than the object's current location, a user has effectively created a stationary object. Thus, the object_attributes type can be instantiated to cover a wide variety of objects, both moving and stationary.

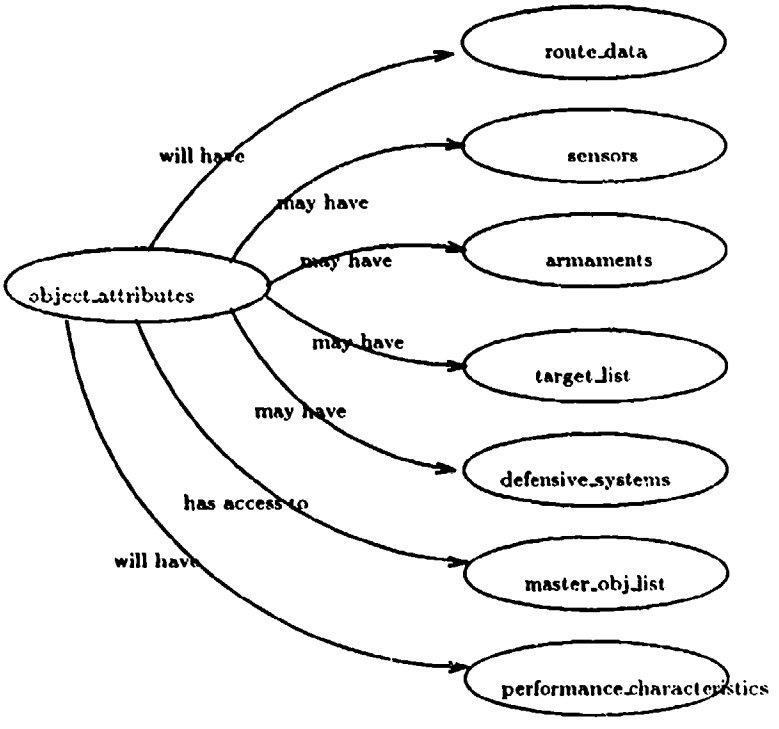


Figure 3.3. Object Relationships

Since each moving object will have some route associated with it, there is a need for the "route_data" object. Even a vehicle that goes nowhere will have a route associated with it, but its route will be a single point. The route data will provide the simulation with the future locations of an object. This information is critical in determining the vehicle's yaw, pitch, and velocity vectors.

"Sensors", like vehicles, can be instantiated with differing attributes, thereby creating different sensors. Many of the vehicles may employ the same type of sensor and some of the vehicles may be equipped with a number of different sensors. Thus sensors logically map to an object class. The association of a sensor, or group of sensors, with a particular object gives the simulation the ability to determine what an object can or will perceive.

The rationale behind the creation of the "armaments" and "defensive_systems" objects is essentially the same as for sensors. The association of a particular type of armament, such as armament type, range, destructive power, etc. with an instantiated object provides the simulation with information which can be used during a fight sequence. Examples of armaments could include systems such as sidewinder missiles, 50 caliber machine guns, or surface-to-air missiles. As with armaments, the defensive systems object provides the simulation with information as to what type of defensive systems an object is equipped, if any. Examples of defensive systems are chaff, flares, or jammers.

The intent of the "operator" object is used to factor in intangible qualities such as experience and threat knowledge. The values assigned to these qualities could be used to help determine whether an attack will be successful (e.g. the armament hit the target). Operator qualities could also be factored into the operator evaluation function where decisions regarding a course of action (such as attack, evade, or take no action) are made.

The "performance_characteristics" object is, as the name implies, where the performance characteristics pertaining to a particular vehicle are stored. The uses of

the information found in this object could be in any calculation needing performance data, especially maximum or minimum limits such as climb, turn or acceleration limits.

The "target_list" object is a linked list containing pointers to each object's targets and each target's location. This information is used in determining if an encountered target should be engaged.

The "master_obj_list" object is a linked list containing pointers to all the objects in the system. Access to this information is critical in the determination of sensor contacts and collision detection.

3.3.2 **The Events** The events are those happenings or occurrences which may cause the system state to change. Events generally cause some process or function to execute which is the driving mechanism which physically changes the system state. As was the case for the objects, the design of the events used in this simulation software calls for a generic approach to their implementation. Again, there are limitations as to how far one can carry a generic approach, but here, too, reasonableness must prevail. As a rule, events should apply equally well to all instantiations of objects within a class (e.g. the reach_turnpoint event should apply equally well to any moving object). Events will be implemented as C functions which in turn will call the applicable functions to make adjustments to scenario object attributes.

Table 3.2 lists the events which are currently used in the simulation work.

The reached_turnpoint event sets a number of functions into motion. As a consequence of reaching a turnpoint, the vehicle's current position is updated. Updating a vehicle's position encompasses five tasks. First, the new position coordinates are transferred from the route data to the current location attributes of the vehicle. Next, the current orientation of the vehicle is calculated and the applicable attributes are updated. Then, the current velocity vectors are calculated, again updating the

Events
reached_turnpoint
entered_sensor_range
made_sensor_contact
collision_distance_reached
ordnance_released
ordnance_reached_target

Table 3.2. **Simulation Events**

appropriate attributes. The object's current time is updated and finally, after all attributes have been updated, the information is sent to the graphics display or to an intermediate file. Once this has been accomplished, the sensor_check function is called to help determine what the next event to be scheduled will be. Ideally, in the absence of any intermediate sensory contacts or collisions (if no sensors are operating), the next event will be the next event point from the vehicle's route data. Thus, in order to determine what the next event really is, a check of the vehicle's projected path must be made against all other paths and positions of stationary objects to determine whether there will be a sensory contact or collision prior to the next predetermined event point. Only then can the proper event be scheduled.

The made_sensor_contact event is basically a decision point. If this event occurs, an object has come within sensory range of another object. The perceiving object at this point must decide what to do about what it perceives. Thus, it must interrogate the source to determine whether it is a friend or enemy, and if it is an enemy, decide on a course action such as attack, evade, or take no action. Thus, the execution sequence is: update the vehicle's position (the same five step process from above), call the operator_evaluation function, schedule the event determined in operator_evaluation, and perform a sensor check to schedule an intermediate event if one is found.

The entered_sensor_range event is identical in *function* to reached_turnpoint.

The position of the object entering the sensor range of another object is updated as above, and then the sensor check function is called to determine what the next event will be for the object in question. Although the subsequent function calls are the same as for both enter_sensor_range and reached_turnpoint, enter_sensor_range is a separate and distinct event caused by sensor range information and the proximity of another object. A reached_turnpoint event, of course, has nothing to do with either of these factors. It should be noted here that for every enter_sensor_range event there should be a corresponding made_sensor_contact event. This makes sense since every time a object senses another object, the other object is coming into sensor range.

The collision_distance_reached event basically means two objects have reached the same point in space at the same time. The collision_distance_reached event will most likely involve a vehicle or vehicles without sensoring capabilities, either because no sensors are present or they are malfunctioning and the vehicle is operating in the blind. As with all other events thus far, the vehicle's position must be updated. A damage_assessment function call would determine the extent of the damage and adjust the appropriate vehicle attribute accordingly. If a total destruction has occurred, then the damage_assessment would also send the graphics display a destruction message signaling that the entity no longer needs to be displayed. At that point the entity will no longer exist within the simulation. In the case of total destruction, damage_assessment will also call the unschedule_events function which will unschedule any event for which the now dead entity was previously scheduled.

The ordnance_released event is scheduled as a result of the operator_evaluation function determining that an attack will take place. The ordnance_released event basically starts a missile on its way to a target. The missile acts as any other moving object in the system, but of course it is moves quite a bit faster. The missile moves along a predetermined route. If it encounters the target before the missile terminates at its last routepoint, an ordnance_reached_target will be scheduled.

The ordnance_reached_target event is scheduled in the sensor_check function

if sensor_check determines the missile catches the object which it is chasing. Ordnance_reached_target updates the position of both missile and target, then it calls hit_miss to determine whether the missile actually scored a hit. In the event of a hit, damage_assessment is called by hit_miss to determine the extent of damage. Since there may be a considerable lag time between the firing or release of the ordnance and its impact, due to the ordnance speed and distance to the target, it is reasonable to model the ordnance impact as a separate event.

3.3.3 Models for Perceive, Move, and Fight

Perceive: The model for perceive deals with how an object becomes aware of another object. Perception takes place through the use of some sensing equipment. Examples of sensing equipment are radar, or the human eye.

The simulation system handles perception between objects by exhaustive comparisons. Typically, before the next predetermined event can be scheduled from a vehicle's route data, the system must determine if an intermediate event needs to be scheduled. Thus the basic model for perceive is given here:

- Compare the vehicle's sensor zone path, from its current location to its next preplanned event location (from its route data), against all other vehicle sensor zone paths or stationary object sensor zone locations.
- Determine if an entered_sensor_range, made_sensor_contact, or a collision_distance_reached will occur prior to the next preplanned event location.
  - If a sensor(s) contact is found, schedule the earliest event.
  - If no contact is found, schedule the next preplanned event from the vehicle's route_data.

Move: Objects move through the system based on information known at the start of the scenario (the route data), and reactions to situations (threats or obstacles). Each vehicle object has, as part of its attributes, route data for the current

scenario. The route data contains the locations of all known events for that vehicle. A typical set of route data will include the location of all turnpoints and targets. Ideally, events are scheduled which coincide with a vehicle moving through turnpoints and targets, eventually arriving at the vehicle's destination. Realistically, vehicles can encounter threats, either ground-based or from another vehicle, or obstacles which may add additional turnpoints to the preplanned route. Thus the basic model for move is iteratively perceive - move (based on perceived data) - perceive.

Fight: Objects from opposing sides may fight if the following conditions are met:

- One or more of the objects is aware of (perceives) another object.
- One or more of the objects is within range of the type of weapon the perceiving object is equipped with.
- The perceiving object has not previously exhausted its armament store.

Vehicles reaching a predetermined target will attack it. Vehicles encountering enemy vehicles or stationary objects from an opposing side may attack based on whether they have extra ordnance allowing them to do so, the probability of enemy destruction versus their own, and whether undetected avoidance is possible.

3.4 Low Level Design

This section details object attributes and the functions which support the occurrence of events. The object structures used in this simulation are in the file sim_stru.h. The code for the functions used in this simulation is in sim_func.h and sim_func.c. These files are listed in Appendix A.

3.4.1 **Object Attributes** The basic construction of the objects are as C structures where the object's attributes are components of the structure. Some of the attributes themselves may also be structures containing additional information.

Thus some nesting of the structures will take place. Figure 3.4 shows the relationship of objects to attributes.

The first object types are those instantiated through the use of the object_attributes structure. Object_attributes can be instantiated to create a myriad of different types of objects as well as creating the same basic type with differing characteristics. A quick glance at the current attributes of this structure may lead one to believe that this structure is used to instantiate only moving objects, since the structure attributes include velocities, rotation rates, etc.... However, moving objects are only a subset of the total item types that can be instantiated using the object_attributes structure. By simply initializing to zero those attributes which are not applicable, the set of non-moving objects can also be created using this structure.

Creating objects through instantiation of a structure is an excellent way to ensure ease of modification and growth of this simulation code. This is because information for new or more complex manipulation of the objects within the simulation can easily be incorporated by simply adding the required attribute to the already existing structure. Below is a brief explanation of the current attributes making up object_attibutes.

Attribute Explanations

int object_type: Used as an icon identifier for the display system

int object_id: Integer value used as a object identifier.

int object_loyalty: Integer value indicating loyalty.

double current_time: The time of the most recent event for the object.

int fuel_status: Integer value denoting the current available fuel.

int condition: Integer value indicating the object's current condition. Values are between 0 and 100, 0 being destroyed, 100 being fully operational.

int vulnerability: Integer value indicating the destructive force needed to destroy the object.

struct location_type location: A structure containing the current location of the

object.

struct xyz_velocities: A structure containing the velocity vectors (v_x, v_y, v_z) of the object.

struct orientation_type orientation: A structure containing the current orientation of the object.

struct rotation_rates: A structure containing the rotation rates around the x, y, and z axis.

struct operator_type operator: A structure containing the operator's qualities such as experience and threat knowledge.

struct performance_characteristics performance: A structure containing the performance characteristics of the object such as the minimum turning radius, max speed, max climb rate, and average fuel consumption.

struct linked_list* sensors: A pointer to a linked list which contains the information about the sensors the object has available to it.

struct linked_list* armaments: A pointer to a linked list which contains the information about the armaments the object has available to it.

struct linked_list* defensive_systems: A pointer to a linked list which contains the information about the defensive_systems the object has available to it.

struct linked_list* route_data: A pointer to a linked list which contains the routing information for the current scenario.

struct linked_list* target_list: A pointer to a linked list which contains information about a object's target(s).

The second object, operator_type, is a structure containing information about the operator's experience and knowledge of the threat. These are two items which are critical to the successful outcome of most confrontations. This object is not being utilized in any of the current simulation algorithms, most notably, the attack sequence algorithm. However, this "hook" was deliberately put in so that this information could be incorporated at a later date to enhance the realism of the simulation.

At this time the operator_type contains the following attributes, but of course it can be expanded if other information becomes necessary.

Attribute Explanations

int experience: An integer value attributable to the operator's experience level.

int threat_knowledge: An integer value attributable to the operator's knowledge of a particular type of threat.

Object three is the performance_characteristics object. This object is basically another hook. It contains some limiting performance factors which could easily be used to determine current fuel status, and whether certain maneuvers are possible. Here, too, more information may eventually be incorporated depending on how detailed performance is modelled.

Attribute Explanations

int min_turn_radius: An integer value giving the minimum turning radius of the vehicle.

int max_speed: An integer value indicating the maximum speed the vehicle could travel.

ave_fuel_cons_rate: A rate indicating how fast the vehicle's fuel is being consumed.

int max_climb_rate: A rate indicating how fast a vehicle could climb (if applicable).

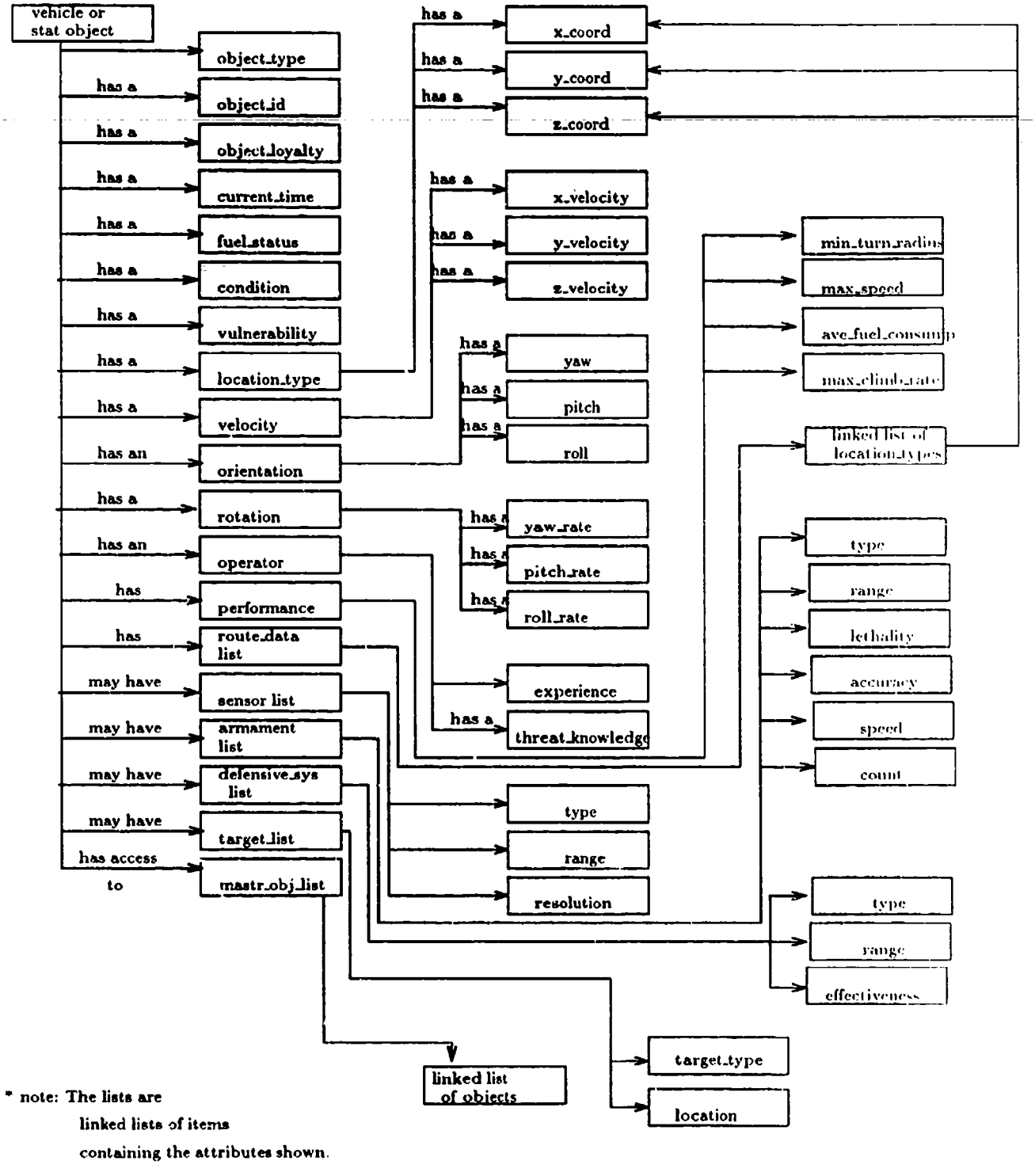


Figure 3.4. Object Attribute Relationships

Object four is a linked list containing the the route points for each object. Every object has its own route data linked list. Even stationary objects will have a route data linked list. The stationary object linked list will contain only one point, it will match the objects current position and will be used to establish the object's position on the display.

Attribute Explanations

struct location_type: Structures which are the x, y, and z, coordinates of the object's route points.

Sensors are object five. Each object can have a linked list containing the sensors available to that object. The attributes are self explanatory. The default value of the function get_sensor_range, if there are no sensors in an object's sensor linked list, is 833 meters, approximately a half mile. The algorithm for sensor selection is presented in the algorithm discussion section.

Attribute Explanations

int type: The integer value which represents a particular sensor such as radar = 1, eye = 2 ...

int range: The integer detection range of the sensor.

int resolution: The integer factor which indicates how clearly an object is seen once detected.

Armaments are object six. Each object may have a linked list of armaments containing the armaments which are available for use by the object. This object has a wealth of information which can be used to add to the realism of the simulation. Currently this object is not being used, but further incorporation of the data contained within this object is straightforward. For instance, the count attribute could be checked and decremented as necessary, before a shot could be allowed.

Attribute Explanations

int type: The integer values which represent a particular type of armament.

int range: Integer value of the range of the armament.

int lethality: Integer value of the destructive power of the armament. Used to determine condition of vehicle or stationary object based on its vulnerability value.

accuracy: Integer value of the accuracy of the armament.

count: How many of a particular type of ordinance are available or left.

Object seven, the defensive_systems, are similar in use to sensors. Each object may have defensive systems which could be used to affect the outcome of a confrontation. Using this information could add to the realism of the simulation. However, at this time this area has been left unaddressed. Attributes could be added to those shown below if necessary.

Attribute Explanations

int type: The integer values which represents a particular type of defensive_system such as chaff = 1, flares = 2, jammer = 3.

int range: Integer value of the range of the defensive_system.

int effectiveness: An integer representation of the defensive system effectiveness.

The target_list is object eight. Each object should have a target list to help determine who the "bad guys" are. The absence of a target list does not mean that an object has no enemies, since a difference in the object_loyalty attribute will indicate whether an encountered object is on the same side or not. Objects without targets will evade other objects without the same loyalty if possible. The usage of the target_list is explained in the algorithm discussion section.

Attribute Explanations

int target_type: The integer value which represents the type of the target (i.e. F15, MIG, TANK ...).

struct location_type: Contains the expected location of the target.

3.4.2 **Supporting Functions** This section gives a verbal description of the functions used to carry out the effects of event occurrences. Object attributes may need to be updated, current and future scenario states may need to be evaluated, and decisions may need to be made. Tables 3.3 and 3.4 shows what functions are used in support of the events possible using this simulation software.

Function: add_event_coords_to_route
Verbal Description: Add_event_coords_to_route uses add_new_routepoint to add a new routepoint to both objects passed as the argument to this function.

Function: add_new_routepoint
Verbal Description: Add_new_routepoint puts a new turnpoint into the route data linked list. The new turnpoint becomes the next prescheduled routepoint.

Function: attack
Verbal Description: Attack creates a "missile" (an instance of object_attributes). Attack initializes the location of the missile, gives it a velocity, creates and inserts three routepoints into the missile's route data linked list. A pointer to the missile is then put into the master_obj_list, and a release_ordnance event is scheduled.

Function: calc_curr_orientation
Verbal Description: Calc_curr_orientation calculates the current orientation of an object based on its current location and its next position from its route data.

Events	*Supporting Functions*	*Supporting Supporting Functions*
reached_turnpoint	update_position	calc_curr_orientation calc_curr_velocities update_object_current_time send_fupdate
	sensor_check	calc_time_at_next_routept get_sensor_range calc_time_at_nextnext_routept line_of_sight difference_in_altitude add_event_coords_to_route add_new_routepoint
entered_sensor_range	update_position	calc_curr_orientation calc_curr_velocities update_object_current_time send_fupdate
	sensor_check	calc_time_at_next_routept get_sensor_range calc_time_at_nextnext_routept line_of_sight difference_in_altitude add_event_coords_to_route add_new_routepoint
made_sensor_contact	update_position	calc_curr_orientation calc_curr_velocities update_object_current_time send_fupdate
	operator_evaluation	get_sensor_range evade attack
	sensor_check	calc_time_at_next_routept get_sensor_range calc_time_at_nextnext_routept line_of_sight difference_in_altitude add_event_coords_to_route add_new_routepoint

Table 3.3. **Events and Supporting Functions**

Function: calc_curr_velocities

Verbal Description: Calc_curr_velocities calculates the current velocity vectors based on its current total horizontal velocity, and its next position from the object's route data.

Function: calc_time_at_next_routept

Verbal Description: Calc_time_at_next_routept calculates the time at the next routepoint based on its current position the distance to the next position and the total velocity vector.

Function: calc_time_at_nextnext_routept

Verbal Description: Calc_time_at_nextnext_routept calculates the time at the routepoint after the next routepoint based on its current position the total distance to the final position and the total velocity vector.

Function: damage_assessment

Verbal Description: The eventual function of damage_assessment is to determine the amount of damage an object has sustained based on vulnerability, current condition, and what ordnance was used. If total destruction has occurred, then call terminate_objects. The current implementation of this function assesses all damage to be total.

Function: difference_in_altitude

Verbal Description: Difference_in_altitude uses the objects current positions, their z velocity vectors, and the time to the next event to determine if the objects will be at the same altitude at the next event time.

Function: evade

Verbal Description: Evade modifies the current velocity, and orientation of the

object in question. Evade also adds a new routepoint to the object's route data and sends the updated poition information to the display driver.

Function: hit_miss

Verbal Description: Hit_miss determines whether the target was hit or missed. This could be based on factors such as range, ordnance accuracy, and defensive systems used, as well as whether the ordnance and target are occupying the same or nearly same location. The current implementation of this function determines hit_miss solely by location of the ordnance and target. If a hit has been determined, damage_assessment is called.

Function: line_of_sight

Verbal Description: The intent of the line_of_sight function is to check to see whether a clear (unobstructed) line of sight exists between two objects. Obstructions may be caused by the terrain or possibly atmospheric phenomena. However, this algorithm remains unimplemented, due in the most part to the fact that terrain has not yet been modelled. The function exists as another "hook", and currently returns true (a valid LOS exists) for all cases.

Function: on_collision_course

Verbal Description: On_collision_course determines whether two objects are on a collision course. The return value is true or false. The primary use of this function is when two objects of the same loyalty encounter each other. Since they are on the same side they don't need to take any action (i.e. attack, or evade), unless they are on a "collision course".

Events	*Supporting Functions*	*Supporting Supporting Functions*
ordnance_released	update_position	calc_curr_orientation calc_curr_velocities update_object_current_time send_fupdate
	sensor_check	calc_time_at_next_routept get_sensor_range calc_time_at_nextnext_routept line_of_sight difference_in_altitude add_event_coords_to_route add_new_routepoint
ordnance_reached_target	update_position	calc_curr_orientation calc_curr_velocities update_object_current_time send_fupdate
	add_new_routepoint hit_miss	
collision_distance_reached	update_position	calc_curr_orientation calc_curr_velocities update_object_current_time send_fupdate
	damage_assessment	

Table 3.4. **Events and Supporting Functions**

Function: on_target_list

Verbal Description: The purpose of on_target_list is to determine whether a threat encountered by an object is an actual target of the perceiving object. The function will return true if the threat is an actual target. The determination algorithm used is explained in the algorithm discussion section.

Function: operator_evaluation

Verbal Description: The basic function of operator_evaluation is to evaluate the threat and choose a course of action. Evaluation of the threat may be in the form of answering questions such as: is the threat a "bad guy", is the threat the intended target, and if it is a friend, are we on a collision course? Courses of action could be attack, evade, or do nothing.

Function: read_data_file

Verbal Description: The read_data_file function is used to read in the initial object data from an ASCII file. The format for this file is shown in Figure 3.5. **Important: Fields are separated by a single blank space, after all required fields are entered for an object (i.e. an F-15) a C compatible End-of-Line (EOL) is entered.**

Function: send_fupdate

Verbal Description: The send_fupdate function is used to send formatted object updates to a datafile which is to be read by a generic display driver. See Appendix D for format and interface requirements of the generic diplay driver.

field 1	field 2	field 3	field 4	field 5	field 6	field 7
int	int	int	double	int	int	int
object type	object id	obj loyalty	curr time	fuel stat	condition	vulnerability

field 8	field 9	field 10	field 11	field 12	field 13	field 14
double	double	double	double	double	double	double
curr x coord	curr y coord	curr z coord	x velocity	y velocity	z velocity	yaw rate

field 15	field 16	field 17	field 18	field 19	field 20	field 21
double	double	int	int	int	int	int
pitch rate	roll rate	experience	threat know	min turn rad	max speed	ave fuel cons

field 22	field 23	field 24	field 25	field 26	field 27	field 28
int	int	double	double	double	int	int
max climb	# routepts	x coord	y coord	z coord	# sensors	sensor type

can be repeated # routept times

can be repeated

field 29	field 30	field 31	field 32	field 33	field 34	field 35
int	int	int	int	int	int	int
sensor range	sensor resolution	# armaments	arm type	arm range	arm yeild	arm accuracy

sensor times

can be repeated # armament times

field 36	field 37	field 38	field 39	field 40	field 41	field 42
int	int	int	int	double	double	double
arm speed	arm count	# targets	target type	targ x coord	targ y coord	targ z coord

can be repeated # target times

field 43	field 44	field 45	field 46
int	int	int	int
# defensive sys	def sys type	def sys range	def sys effect

can be repeated # defensive sys times

Figure 3.5. Input File Format

Function: sensor_check

Verbal Description: Sensor_check compares a object's projected sensor zone path with all other sensor zone paths and positions of stationary sensor zones within the system to determine if the object's sensor will pick up anything before its next predetermined scheduled event. In order for a sensor to be able to "see" another object the following rules should be satisfied:

- The sensor being used must be operational.
- The object to be detected must move within the sensor's range.
- An unobstructed Line-Of-Sight (LOS) must exist between the sensor and the object being sensed. An unobstructed LOS is dependent in part on which sensors are being used.

Five different events can be scheduled by sensor_check depending on what is found during the sensor_check evaluation. If a sensor contact is found, and the sensor range of both objects in question is zero, along with an altitude seperation of less then five meters, a collision_distance_reached event will be scheduled. If a sensor contact is found but there is a sensor range being used greater then zero or there is a difference in altitude, then either an entered_sensor_range, made_sensor_contact, or (in the case where the object is a missile) an ordnance_reached_target will be scheduled. If a no contact is found, then a reached_turnpoint event is scheduled at the appropriate time. The algorithm used to implement this function is detailed in the algorithm discussion section.

Function: terminate_objects

Verbal Description: The terminate_objects function sends a message to the display file indicating that an object need not be displayed any longer. It deletes all currently scheduled events from the next event queue involving the now dead object, deletes

the object pointer from the master_obj_list, and frees the memory used to hold the event_argument.

Function: update_object_current_time

Verbal Description: Update_object_current_time simply assigns the event_time to the object's current time attribute, thus updating the object current time to the current event time.

Function: update_position

Verbal Description: Update_position takes the next route point from the route data and updates the current location of the object. It then calls in this order, calc_curr_orientation, calc_curr_velocities, update_object_current_time, and send_fupdate.

3.5 The Interfaces

3.5.1 **Overall System Interface** In keeping with the modularity and object construction design scheme, the code has been constructed to facilitate modification and growth. The overall system interfaces are illustrated in detail in Figure 3.6. It shows a rather complex structure which makes use of two generic C packages, the generic simulation package (sim_driv.h and sim_driv.c) and the generic linked list package (ll.h and ll.c), which were developed in a separate effort and are given in Appendix C. The simulation structures, simulation events, supporting simulation functions, and main simulation code are all in separate files and are provided as Appendix A. The only other interface is the generic display driver interface. The generic display driver was a separate, but concurrent research effort (9). The interface requirements are provided in Appendix D.

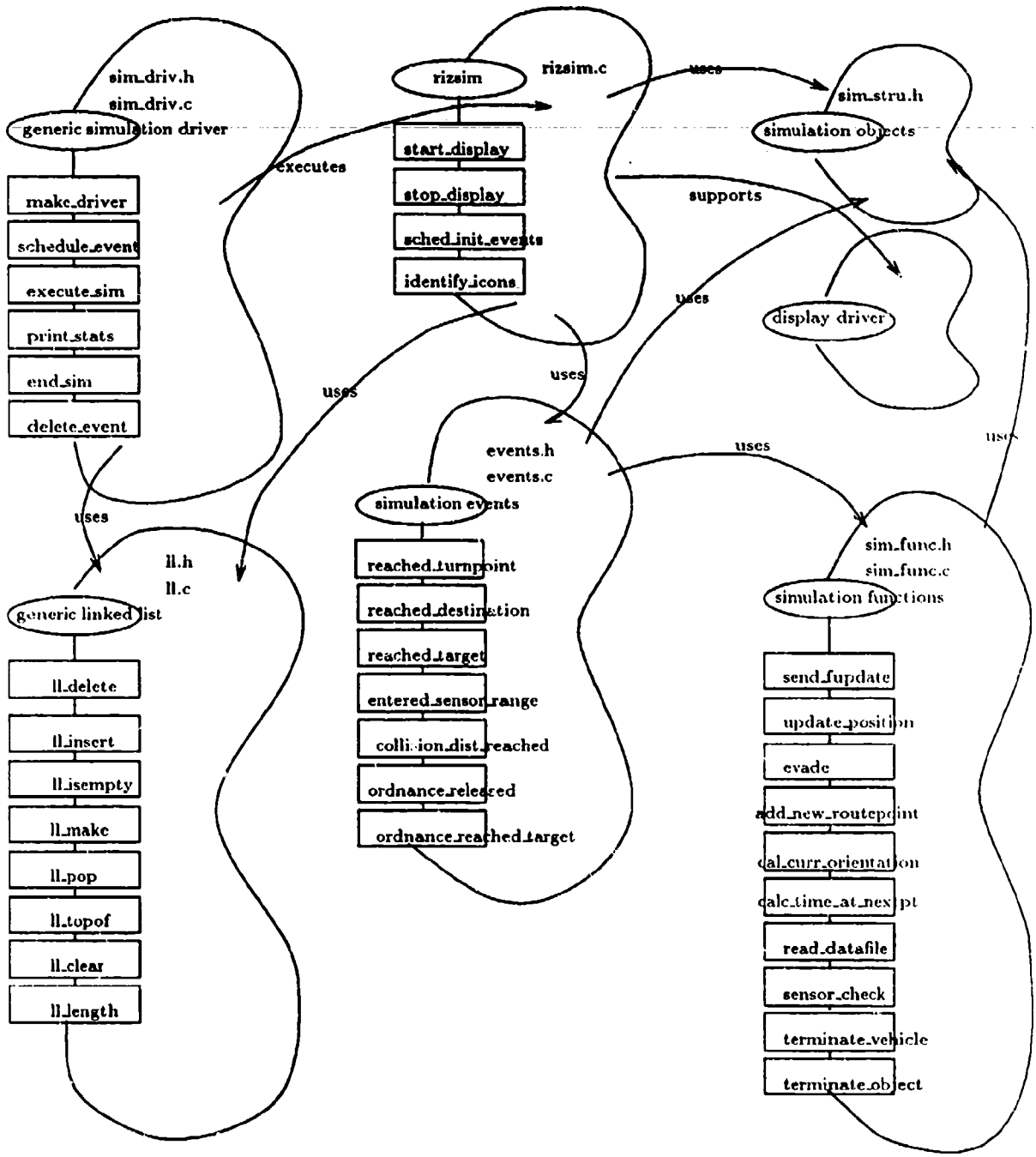


Figure 3.6. Overall System Interfaces

3.5.2 **The Driver Interface** As was the case in the areas discussed previously, the simulation driver was approached generically. The design will not be covered in detail here since it was completed as separate work and is given in Appendix C. The overall function of the driver is to execute the simulation. It accomplishes this through the use of the functions ***make_driver, schedule_event, execute_sim, delete_event,*** and ***end_sim.*** These functions are all available to the user writing an event driven simulation which makes use of a NEQ.

The following are brief explanations of the functions of the generic simulation driver.

make_driver: The make_driver function allows the user to create an instance of the simulation driver. The user can then use the other simulation driver functions available to manipulate the the driver in creating a working simulation. A comparison function is supplied by the user to the driver to allow the driver to properly sort events.

schedule_event: The schedule_event function allows the user to schedule events by passing a pointer to the event function, its arguments, and the time of the event with the simulation identifier 'driver'.

execute_sim: The execute_sim function executes the functions(events) which have been scheduled with the schedule_event function. Execute_sim will continue dispatching events until there are no more events scheduled in the NEQ.

delete_event: The delete_event function gives the user the ability to remove previously scheduled events from the NEQ. Using the event_id, returned to the user when using "schedule_event", delete_event searches for a matching event_id in the NEQ and deletes it.

end_sim: The end_sim function gives the user the ability to stop the simulation. End_sim effectively empties the NEQ.

IV. MAJOR ALGORITHM DISCUSSIONS AND IMPLEMENTATIONS

The following sections highlight the major algorithms used in the simulation implemented as part of this thesis work. Although each function has an associated algorithm only those deemed in need of a more detailed explanation are given here. These algorithms represent the more complex or more interesting algorithms of the simulation code. These algorithms are simply one way to model these functions. They could, and possibly should, be modified to create more realism in the simulation.

4.1 The Evade Algorithm

The basic high level algorithm employed is straightforward. However its implementation is considerably more complex due to the number of special cases which exist. The basic algorithm states:

- Calculate or determine the threat object's path.
- Calculate and adjust the evading object's orientation and velocity vectors such that the new direction is 90 degrees from the threat path, moving away from the threat path.
- Calculate a new routepoint for the evading object, given the evading object's new orientation and velocity vectors.
- Add the new routepoint to the evading object's route data.

There are three special cases which must be handled separately due to the usage of trigonometric functions and divide by zero problems.

- Case I: When the x velocity vector of the object to be evaded equals zero, and the y velocity of the object to be evaded is not equal to zero. This is the situation in two dimensions where the object to be evaded is moving on a path which is parallel to the y axis. If this situation exists, then according to the algorithm, the direction of evasion will be along a path parallel to the x axis. What now must be established is whether the movement will be in the positive x direction or the negative x direction. This is established by simply evaluating the difference between the x coordinates of the two objects. Once the direction is known, the total velocity vector is then applied to that direction. This situation is illustrated in Figure 4.1.

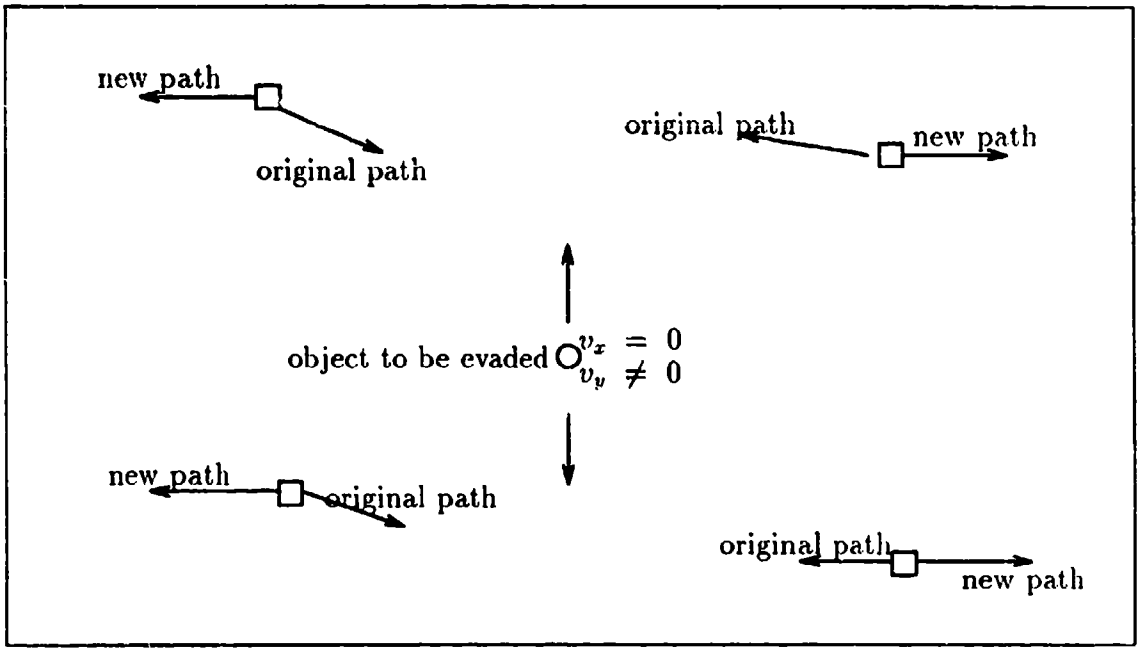


Figure 4.1. CASE I: x velocity vector = 0, y velocity vector $\neq$ 0

- Case II: Case II is just the opposite of Case I. Here the x velocity vector of the object to be evaded is not equal to zero and the y velocity vector is equal to zero. Handling this situation is logically identical to Case I so it need not be detailed again. This situation is illustrated in Figure 4.2.

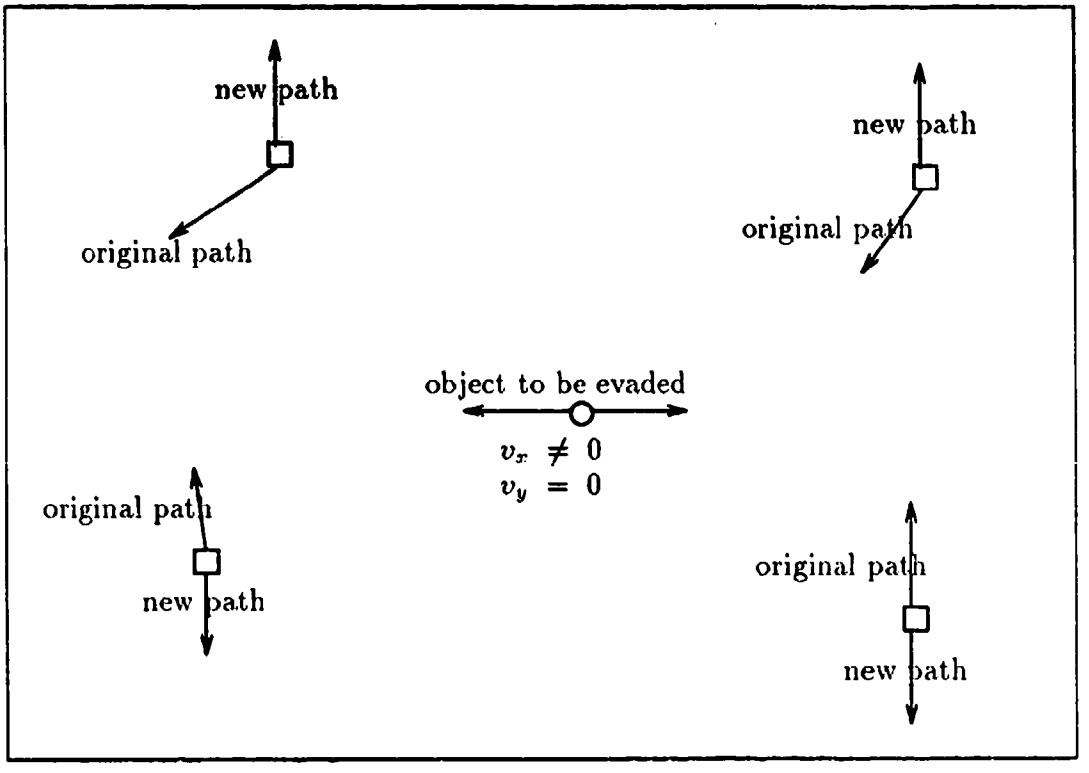


Figure 4.2. CASE II: x velocity vector $\neq 0$, y velocity vector $= 0$

- Case III: Case III is slightly more tricky. In Case III both the x and y velocity vectors of the object to be evaded are zero. This means that the object to be evaded is stationary, and as a consequence will also *not* be eventually moving out of the path of the other object. So how then does the evading object get by the object to be evaded? The algorithm for this situation is as follows and is illustrated in Figure 4.3:

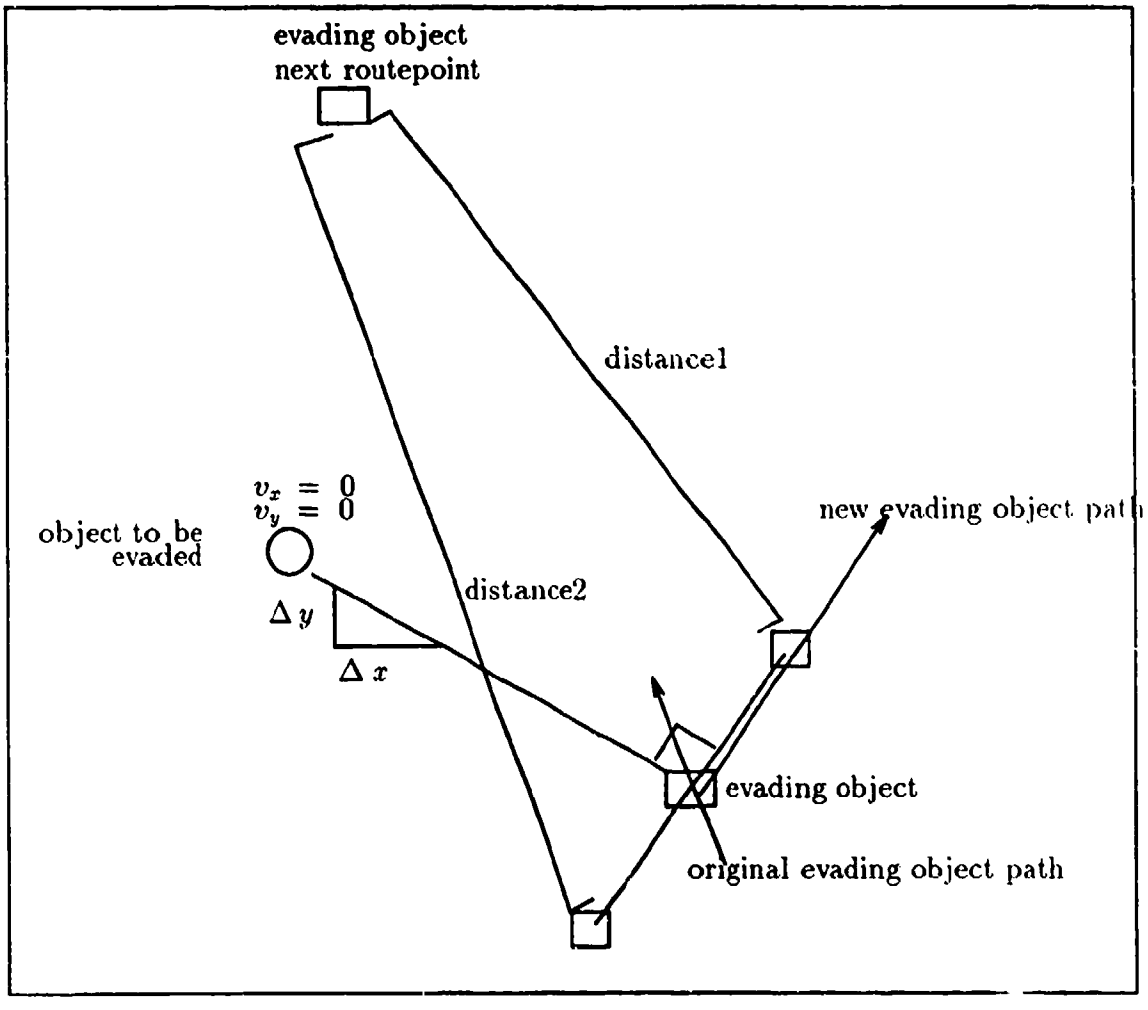


Figure 4.3. CASE III: x velocity vector = 0, y velocity vector = 0

- Draw a line between the current locations of the two objects and calculate the slope of this line.
- Calculate two new locations. Each location should be an equal distance and on opposite sides of the evading object along a line which is 90 degrees from the line found in step one.
- Now, compare the distances from each new point to the evading object's next routepoint. The shorter of the two distances indicates the proper direction for the evading object to turn.
- Add the new routepoint to the evading object's route data.
- Calculate the current orientation of the evading object.

There is one special case within this case which also warrants mentioning. This situation occurs when an object's next preplanned routepoint is within its own sensor range of the object it is trying to evade. When this occurs, the evading object would, after moving to an intermediate evasion point, try to return to its next preplanned routepoint which it could never get to, since evade would simply be called over and over again. Thus, the way this is handled is that before the evasion point is calculated and loaded into the evading object's route data, the next preplanned routepoint is checked to see if it is usable (not too close to the object to be evaded). If it is too close to the object to be evaded, then that point is discarded and the next preplanned routepoint takes its place.

In the general case, both objects are moving and the object to be evaded is not moving in a path parallel to any axis. The algorithm for this case is as follows:

- Calculate the slope of the object to be evaded using its x and y velocity vectors.
  - Assign the negative inverse of this slope to the slope of the evading object.
  - Using the standard equation of a line, $y = mx + b$, calculate the y intercepts of both lines.
  - Simultaneously solve both equations for a common x and y coordinate.
  - The difference between the common x and y coordinates and the current x and y coordinates of the evading object indicates the proper sign of the x and y velocity vectors of the evading object.
  - Calculate the slope angle, *atan* (*evader slope*).
  - The magnitude of the x and y velocity vectors of the evading object will be the absolute value of the total velocity vector times, the cosine of the slope angle and the sine of the slope angle respectively.

This algorithm is illustrated in Figure 4.4.

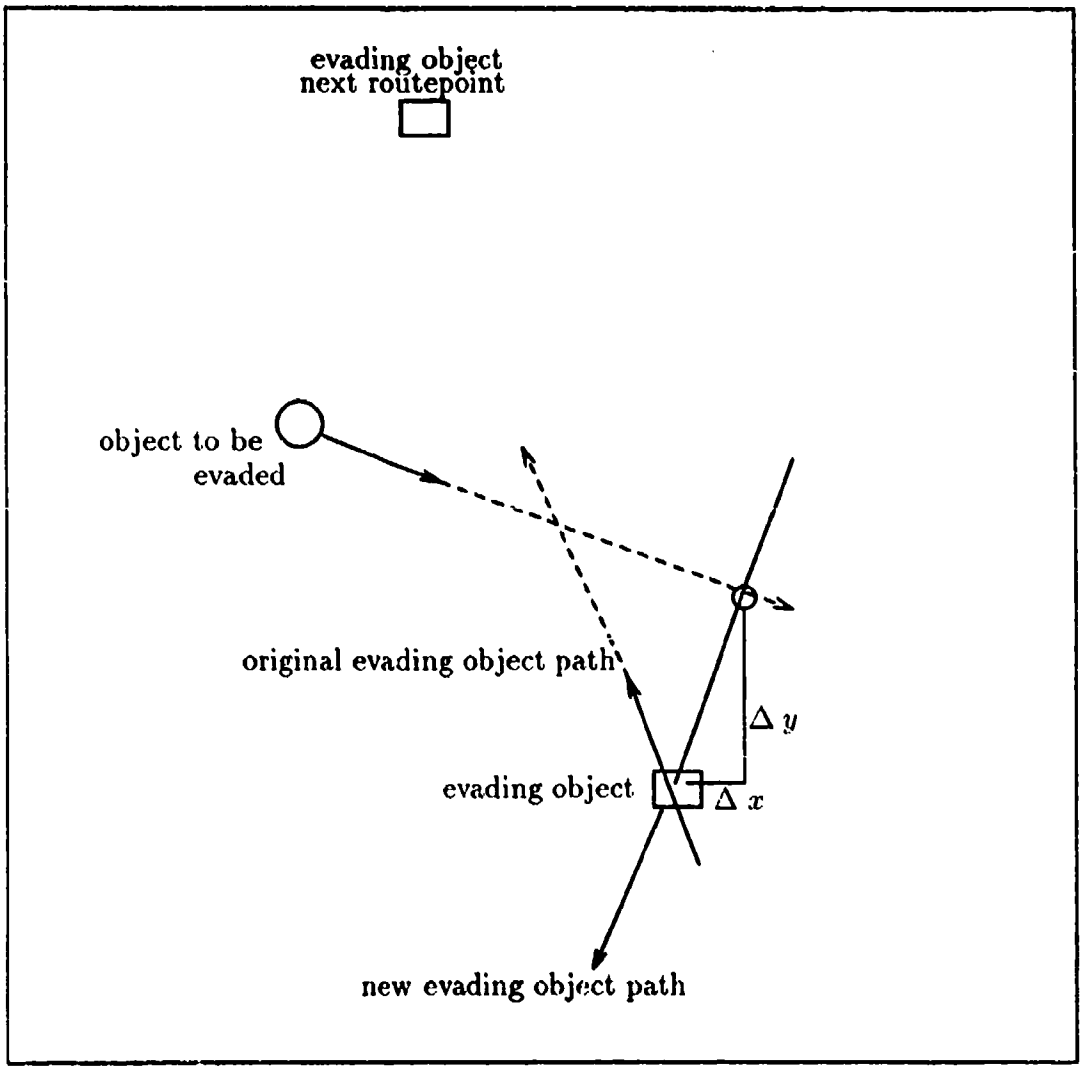


Figure 4.4. GENERAL CASE: x velocity vector $\neq 0$, y velocity vector $\neq 0$

4.2 The Sensor Check Algorithm

The sensor_check routine creates the majority of the workload for the hardware running the simulation. This is because every time the sensor_check routine is called, every object in the simulation scenario must be interrogated. Although this routine creates a large workload, it does not necessarily mean it is the most complex of the algorithms used in this simulation software. Indeed, the high level algorithm is easily understood.

- Before scheduling an object's next preplanned event, determine if there are any other events which should take place prior to the preplanned event; schedule the earliest event.

At the heart of this algorithm is the quadratic equation (31). Specifically, it is the solution of the quadratic equation in (t) which yields the sought after time at which two objects will come within a given distance (d) of each other. The usage of the quadratic equation in (t) is illustrated here. If two moving objects have a current position of (x_{at_1}, y_{at_1}) and (x_{bt_1}, y_{bt_1}) as shown in Figure 4.5, then their respective coordinates at future time (t) can be represented by the following equations:

$$x_{at} = x_{at_1} + v_{xat_1}(t - t_1)$$

$$y_{at} = y_{at_1} + v_{yat_1}(t - t_1)$$

$$x_{bt} = x_{bt_1} + v_{xbt_1}(t - t_1)$$

$$y_{bt} = y_{bt_1} + v_{ybt_1}(t - t_1)$$

x_{at} is the x coordinate of object "a" at some time t
x_{at_1} is the current x coordinate of object "a" at time t_1
v_{xat_1} is the current x velocity vector of object "a" at time t_1
y_{at} is the y coordinate of object "a" at some time t
y_{at_1} is the current y coordinate of object "a" at time t_1
v_{yat_1} is the current y velocity vector of object "a" at time t_1
This assumes v does not change between t and t_1

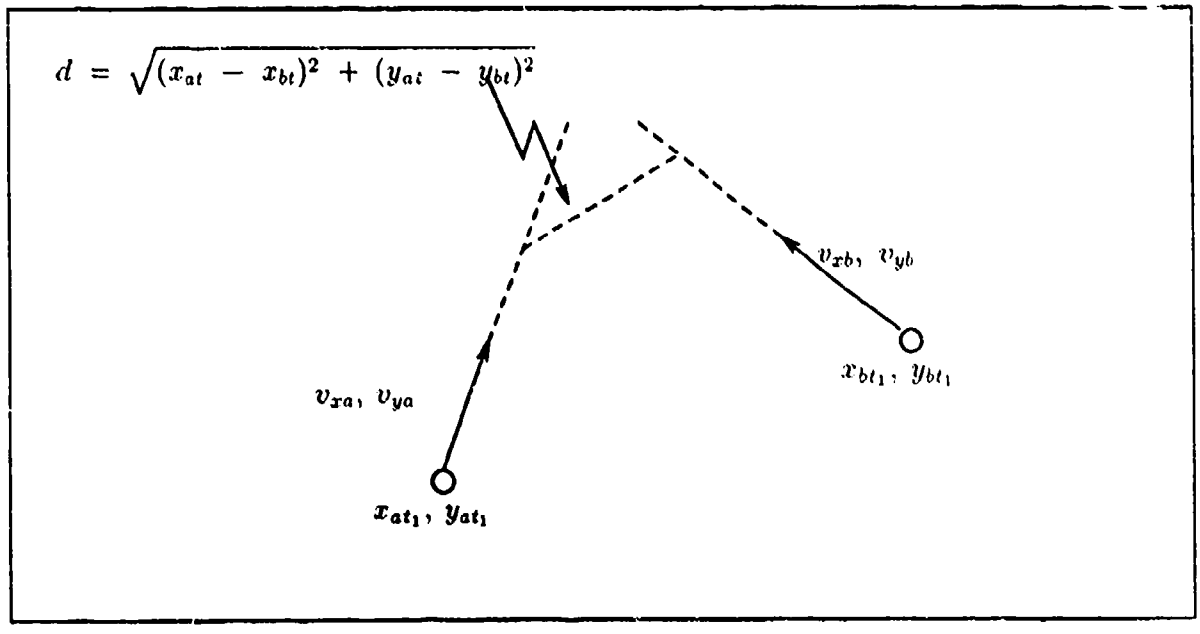


Figure 4.5. Illustration of Calculation

Now if we let: $x_{at_1} = X_A$ and $v_{xat_1} = V_{XA}$ and $y_{at_1} = Y_A$ and $v_{yat_1} = V_{YA}$ etc. and $\Delta t = t - t_1$ (the time until the event will occur, e.g. if Δt is 5, then the event will occur in five time units from the current time). Then

$$x_{at} - x_{bt} = (X_A + V_{XA}\Delta t) - (X_B + V_{XB}\Delta t) = (X_A - X_B) + (V_{XA} - V_{XB})\Delta t$$

Making similar substitutions for the y coordinates yield

$$y_{at} - y_{bt} = (Y_A - Y_B) + (V_{YA} - V_{YB})\Delta t$$

According to the distance formula: $d(t) = \sqrt{(x_{at} - x_{bt})^2 + (y_{at} - y_{bt})^2}$ Making the substitutions into the distance formula yield:

$$((X_A - X_B) + (V_{XA} - V_{XB})\Delta t)^2 + ((Y_A - Y_B) + (V_{YA} - V_{YB})\Delta t)^2 = d^2$$

For clarity let, $l = (X_A - X_B)$, $m = (V_{XA} - V_{XB})$, $n = (Y_A - Y_B)$, and $p = (V_{YA} - V_{YB})$. Putting it into the form of the quadratic equation yields:

$$(m^2 + p^2)\Delta t^2 + (2lm + 2np)\Delta t + (l^2 + n^2) - d^2 = 0$$

$$\Delta t = \frac{-(2lm + 2np) \pm \sqrt{(2lm + 2np)^2 - 4(m^2 + p^2)(l^2 + n^2 - d^2)}}{2(m^2 + p^2)}$$

The distance (d) can be varied according to the range of the sensor being used by the objects in question. It should be noted here that this application of the quadratic equation is in only two dimensions, thus the sensor zones of any object appear as a cylinder that knows no bounds in the z direction as shown in Figure 4.6. Non-imaginary solutions to the quadratic are contact points, assuming the objects

actually progress along their current routes without change. Imaginary solutions indicate that an object will not intersect another object's sensor zones , or the object is already within the sensor zone area of the other object. The two solutions that can be found are the time at which the an object encounters a zone and the time at which an object exits a zone. This implementation of sensor_check uses only the first solution, the time entering the zone.

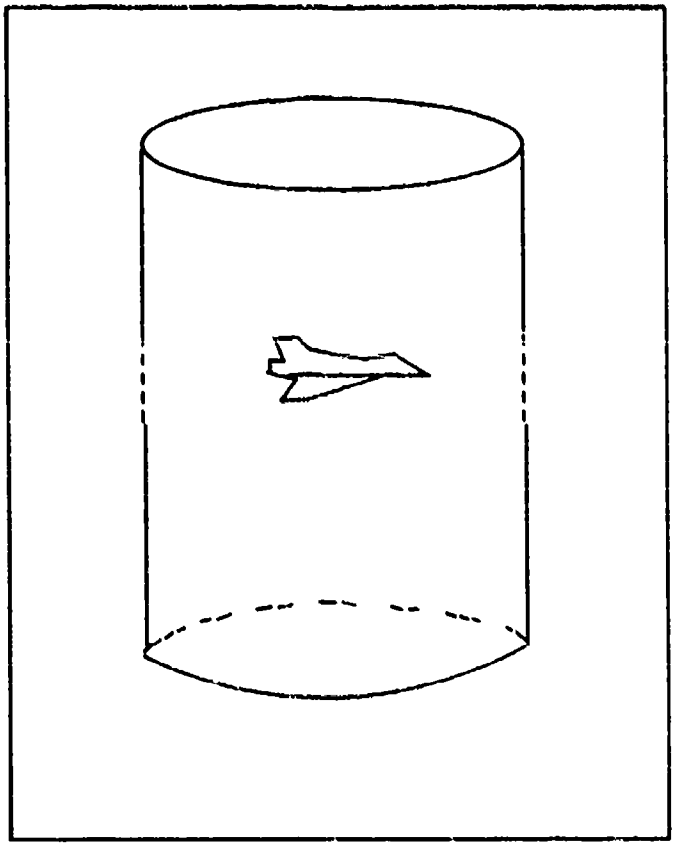


Figure 4.6. Illustration of Object Sensor Zone

The actual implementation is not as straightforward as the original algorithm implies. The implementation algorithm follows.

```
-For all moving objects
    -For as many objects as there are in the master_obj_list
        -While the popped object's id is not = to the current object id
            -Calculate the term under the radical of the quadratic equation solution
             using both the current object's and the popped object's sensor range
                -Calculate the time the popped object will reach its next preplanned event
                -If term under radical in solution is >= 0 using the current object's sensor range
                    -Calculate the sensor contact time1 (the quadratic equation solution)
                    -If the sensor contact time is < the event time and > the current time
                      and <= the time of the other object's next scheduled event, and <= the time
                      of its own next preplanned event, and a line of sight exists between objects.
                        -Set the valid_contact1 flag to TRUE
                -If the term under the radical >= zero using the other object's sensor range
                    -Calculate the sensor contact time2 (the quadratic equation solution)
                    -If the sensor contact time is < the event time and > the current time and
                     <= the time of the other object's next scheduled event, and <= the time of
                     its own next preplanned event, and a line of sight exists between objects.
                        -Set the valid_contact2 flag to TRUE
            -If valid_contact1 or valid_contact2 are TRUE
                -Set the appropriate contact flag(s) to TRUE or FALSE
    -end for
-end for
-If contact1 and contact2 are TRUE, and sensing range = 0, and difference in altitude = 0
    -Schedule a collision event
-Else if contact1 or contact2 is TRUE
    -Add the appropriate routepoints to the current object's route data
    -If contact2 is TRUE
        -Schedule an entered_sensor_range event
    -If contact1 is TRUE and the current object is not a missile
        -Schedule a made sensor contact event
    -if contact1 is TRUE and the current object is a missile
        -Schedule an ordnance_reached_target event
-Else
    -Schedule a reached_turnpoint event for the current object
```

4.3 The Operator Evaluation Algorithm

The operator_evaluation algorithm is a very simple decision tree which culminates in a course of action, either do nothing, evade, or attack. The basic decision tree is depicted in Figure 4.7.

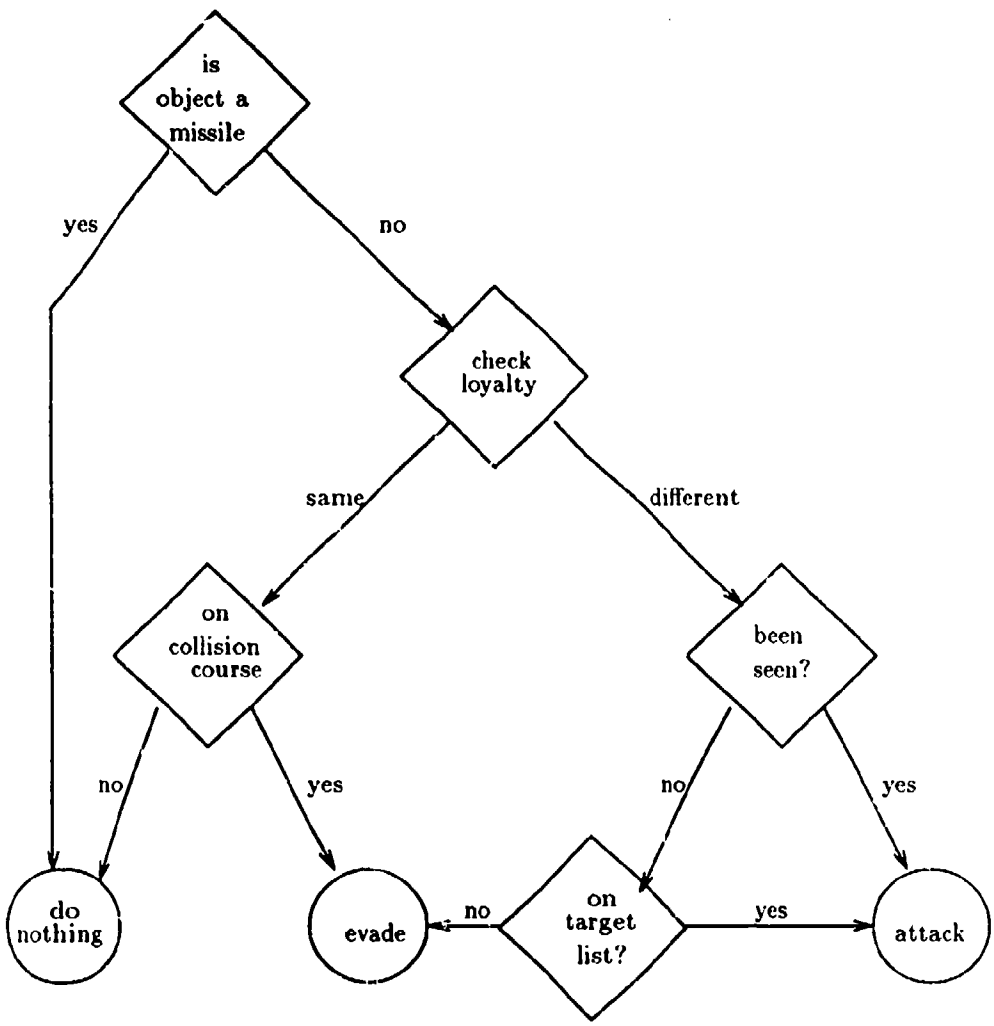


Figure 4.7. Decision Tree for the Operator Evaluation Algorithm

Although rudimentary, this algorithm could be considered a very simple expert system where an operator's thought process is being modelled. The possibilities for expansion to this algorithm are limitless.

4.4 The Attack Algorithm

The attack algorithm says:

> Instantiate a MISSILE object
> Initialize the MISSILE'S attributes
> Determine and load the MISSILE'S routepoints
> Fire the MISSILE

The approach taken in determining a missile's route was to give each missile three routepoints. The first routepoint for the missile would be the current location of the object from which it is being launched. The second routepoint will be the current location of the object at which it is being fired, keeping in mind that a moving target will continue moving from its current location. Thus, a third routepoint must be included if the missile is to have any chance of hitting its target. The third routepoint of the missile is set to the target's next scheduled routepoint. Therefore, although not occurring often unless scripted in that way, a target object can "get away" if it can reach its next routepoint before the missile catches it. In the event that the missile does not catch the target, the missile simply dies at that point.

4.5 The Update Position Algorithm

The update_position algorithm is the most used algorithm in this simulation. It is used in conjunction with other algorithms in every event in this simulation software. The code for this algorithm is compact, making use of four function calls

from within update_position. The order of the function calls is critical to ensure calculated values are correct. The algorithm is stated here:

Pop the next position from the object's route data

Assign the next position coordinates to the current position coordinates

Calculate the current orientation of the object

Calculate the current velocity vectors of the object

Update the object's current time

Send the update to a file (or directly to the display driver, if available)

If there were no routepoints left on the list and the object was a MISSILE

Terminate the missile

4.6 The Add New Routepoint Algorithm

The add_new_routepoint algorithm is very simple, given some time in the future at which the object is to arrive at the new point (e.g. add a new routepoint at a time 30 seconds from now). Thus the algorithm reads:

To the current x, y, and z coordinates, add the time to the new routepoint multiplied by the respective velocity vectors.

Add the newly calculated point to the object's route data.

For example, if the current x, y, and z coordinates were 100, 250, and 1000, with v_x, v_y, $and\, v_z$ as 200, 200, 0, then new route point x, y, and z coordinates at a time 15 seconds from the current time would be, 100 + (200)(15), 250 + (200)(15), and 1000 + (0)(15).

There is one critical factor involved in the determination of a new routepoint. That fact is, the object must have its velocity vectors properly adjusted to reflect

the new direction of movement **before** the new routepoint can be calculated. Obviously this must be so since using the wrong velocity vectors will yield incorrect new coordinate values.

4.7 The Calculate Current Orientation Algorithm

The calc_curr_orientation algorithm uses standard trigonometry to find the angles between the object's current coordinates and the object's next routepoint. Thus the basic algorithm used was:

> Pop the next routepoint from the object's route data.
> Calculate the angles between the two points, yaw and pitch
> (roll is not calculated in this implementation).
> Adjust the object's attributes accordingly.
> Reinsert the popped routepoint into the object's route data.

4.8 The Calculate Current Velocities Algorithm

The calc_curr_velocities algorithm uses a similar approach to calc_curr_orientation although the trigonometry is slightly more involved. The algorithm for this function is:

> Pop the next routepoint from the object's route data.
> Calculate the total horizontal velocity vector.
> Find the angle between the current location and the next routepoint
> location.
> Calculate the horizontal distance between the two points.
> Using the distance and horizontal velocity vector, calculate
> the time to the next routepoint.
> Using the sine, and cosine of the angle found, calculate the new

x and y velocity vectors.

Using the delta z value and the time to the next routepoint, calculate the new z velocity vector.

Reinsert the popped routepoint into the object's route data.

4.9 Other Algorithms

The following algorithms were not discussed in this chapter because they are believed to be easily enough understood through their respective module headers and by simply stepping through the actual code. The actual code is in Appendix A.

send_fupdate

calc_time_at_next_routept

calc_time_at_nextnext_routept

read_datafile

terminate_vehicle

get_sensor_range

line_of_sight

damage_assessment

hit_miss

difference_in_altitude

add_event_coords_to_route

update_object_current_time

on_collision_course

on_target_list

V. RESULTS, CONCLUSIONS, RECOMMENDATIONS

5.1 Results, Meeting the Objectives

At the onset of this thesis effort, time was spent defining basic objectives which needed to be met. These objectives were found, of course, to be driven by the planned usage of the final product. That plan was to parallelize the simulation code created by this thesis and to use the parallel version in speedup studies concerning military simulation executions on parallel computers. Thus the following were the objectives defined:

1. Create a military scenario simulation using a modular object-oriented design.
2. Use the C programming language.
3. The final product must be easily modified.
4. The final product should exhibit a high degree of computational complexity.
5. The simulation code should be generic in nature such that differing scenarios could be run simply by altering the input data.
6. The simulation output should interface with the generic display driver developed by DeRouchey (9).

It is believed that these objectives have been met. C structures were used to create objects and their attributes. This design enhances both the modularity and object-oriented nature of the simulation code. Structures of this nature ensure that all the information regarding the object are always physically tied to that object and can be found, used, or modified, by making the correct reference to the instantiated structure. The usage of these types of structures also gives the simulation code much of its flexibility and growth potential. Adding to, or changing the existing attributes of any of the structures within the simulation can make available more or different

information which in turn could be used to increase the complexity and/or realism of the simulation. Adding to the ability of the code to be easily modified is the overall structure of the files, their interconnections, and the generic structure of much of the code. For instance, the simulation structures are in a separate file, easily found, and easily modified. The same is true for the simulation functions. The simulation driver is also a separate package, as is the linked list code. Either package could be replaced if it were desired. These packages were also created using a fairly strict modular object-oriented design, thus enhancing their modification potential. So, not only does the simulation lend itself to modification, but so does its associated code, in part or as a whole. Having stated the above, it is easy to see that objectives one and three have definitely been met. Computational complexity is reached in this simulation in two ways. First, there is the nature of the calculations themselves. This simulation makes use of numerous computations involving long float valued numbers. Operations on these numbers include addition, subtraction, division, sine, cosine, tangent, arctangent, and square root. The second area of computational load comes from the sheer number of times these operations are required. The bottom line is that computational load can be increased simply by adding objects to the simulation scenario. Thus, objective four can be put to rest. There isn't much to say about objective five. The implementation allows the flexibility to create a nearly unlimited number of scenarios and thus unlimited simulations.

5.2 Conclusions

It was never an objective of this work to create an accurate military simulation. Once the design phase began in earnest, it became readily apparent that had accuracy been a requirement, the amount of work required would far exceed what could be accomplished by one person in one thesis cycle. Realistic military simulations can take teams of programmers and modelers years to produce (13). Indeed, the creation of a "representative" military simulation proved to be no easy task. The complexity

of the simulation grew quickly as the possible execution paths increased with every implementation of a new function. In fact, trying to predict all the events of simulations involving more then five interacting objects prior to the actual execution becomes extremely difficult, if not impossible. Once the execution is complete, verification of what actually occurred is somewhat easier. However, stepping through the output can be a lengthy process. By far the best way to verify the output is to view the output via the graphical display driver discussed earlier. It should be pointed out, however, that viewing the output does not necessarily mean that all events were scheduled properly. Since the display driver operates on a principle of extrapolation, an object's position will continue to be updated even if an event is somehow omitted. The best approach to running a verifiable simulation is to first script the simulation as completely as possible; second, try and verify a printout of the output file against the script; then view the simulation's graphical display to check the overall correctness of directions of movement, relative speeds, kills, and pitch angles.

It is felt that the work of this thesis effort represents only the skeleton of what a real military simulation could ultimately look like. Addressing the basic areas at least in some way, even if only as a hook, represents a significant part of the overall effort. It is the opinion of the author that this is probably the most difficult part of the total effort. What lies ahead, prior to parallelizing, is enhancing the code, and adding realism. This part is the "putting the meat on the bones" part.

5.3 Recommendations

Recommendations generally fall into two catagories, either those concerned with enhancing the serial version of the simulation code, or those concerned with the parallelization issues. As far as enhancing the serial simulation code, the possibilities are almost endless, depending on the level of detail sought. Listed here are just a few of those possibilities:

- Modify the sensor check algorithm to include the third dimension. This would give objects a spherical sensor zone instead of cylindrical. By off-setting the actual object location from the center of the sensor zone, one can create a sensor zone in front of, behind, above or, below an object.
- Use more of the existing object attributes, such as weapons and operator attributes, to add more realism.
- Add more nondeterminism into the detection and attack processes.
- Add more error checking into the input function. Although each input value cannot be verified correct, they can be checked to be within acceptable ranges.
- Terrain still needs to be addressed. This also creates a need for a viable implementation of a Line of Sight function.
- Modify the damage assessment function to include damage less then total destruction.
- Add some expert type decision making in choosing of sensor and/or armaments for an object to use.
- Add some nondeterminism to the hit miss function.

Parallelization is a separate issue. The issues here are what machine to use, distributed or shared memory, and how to partition the simulation. Suggestions to these questions follow:

- Using the shared memory machine would alleviate problems associated with global data which may make things a little easier to handle.
- If the choice is made to use a distributed memory machine, it is suggested to break the problem space up into areas of set dimensions. Then create an artificial event type called "reached_node_boundary" to represent the point in time that an object needs to be handed over to another node.

Appendix A. SIMULATION CODE

A.1 Simulation Structures

The following is a copy of the simulation structures file, sim_stru.h.

```
struct location_type
{
double x_coord;
double y_coord;
double z_coord;
};

struct xyz_velocities
{
double x_velocity;
double y_velocity;
double z_velocity;
};

struct orientation_type
{
double roll;
double pitch;
double yaw;
};

struct rotation_rates
{
double roll_rate;
double pitch_rate;
double yaw_rate;
};

struct operator_type
{
int experience;
int threat_knowledge;
};

struct performance_characteristics
{
int min_turn_radius;
int max_speed;
int ave_fuel_cons_rate;
int max_climb_rate;
};

struct sensors
{
int type;
int range;
int resolution;
};

struct armaments
```

```
{
int type;
int range;
int lethality;
int accuracy;
int speed;
int count;
};

struct defensive_systems
{
int type;
int range;
int effectiveness;
};


struct targets
{
int target_type;
struct location_type target_location;
};


struct event_args
{
double event_time;
struct object_attributes *object1;
struct object_attributes *object2;
};



struct object_attributes
{
int object_type;
int object_id;
int object_loyalty;
double current_time;
int fuel_status;
int condition;
int vulnerability;
struct location_type location;
struct xyz_velocities velocity;
struct orientation_type orientation;
struct rotation_rates rotation;
struct operator_type operator;
struct performance_characteristics performance;
struct linked_list *route_data;
struct linked_list *sensors;
struct linked_list *armaments;
struct linked_list *defensive_systems;
struct linked_list *target_list;
};
```

A.2 Rizsim Code

The following is a copy of the actual simulation code, rizsim.c.

```
#include "ll.h"
#include "sim_driv.h"
#include "sim_func.h"
#include "sim_stru.h"
#include "events.h"
#include <stdio.h>
#include <malloc.h>

/*****************************************************************************/
/* DATE: 08/02/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: The main simulation code. The source of the simulation run         */
/* FILENAME: rizsim.c                                                        */
/* COORDINATOR: Rob Rizza                                                    */
/* PROJECT: MS Thesis GCS-90D                                                */
/* OPERATING SYSTEM: MS-DOS                                                  */
/* LANGUAGE: Microsoft Quick-C                                               */
/* FILE PROCESSING: Link and Compile with sim_driv.c, sim_func.c, events.c,  */
/*                  and ll.c                                                 */
/* CONTENTS: see prototypes in next section                                  */
/* FUNCTION:  Basically, this code intiiates the execution of the simulation*/
/*****************************************************************************/

/*****************************************************************************/
/*                 PROTOTYPES OF FUNCTIONS WITHIN RIZSIM.C                   */
/*****************************************************************************/
void start_display ();
void stop_display (); /* doubl, last_event_time */
void schedule_init_events ();
void identify_icons ();
int compare_time (); /* double* time1, double* time2 */

/*****************************************************************************/
/*                       GLOBALS USED IN RIZSIM.C                            */
/*****************************************************************************/
struct linked_list *master_obj_list;
struct driver *simulation_driver;
int highest_obj_id = 0;

/*****************************************************************************/
/*                  RIZSIM.C MAIN CODE BEGINS HERE                           */
/*****************************************************************************/
void main ()
{
struct linked_list *stats_queue = NULL;
struct driver_data *last_event = NULL;
double last_event_time;

struct driver_data *deleted_event;
struct linked_list *deleted_event_list;

identify_icons ();

simulation_driver = make_driver (8, compare_time);

master_obj_list = ll_make (FIFO);

read_datafile ("datafile.c");

schedule_init_events ();

start_display ();

stats_queue = execute_sim (simulation_driver);
```

```
last_event = (struct driver_data*)ll_pop (state_queue);
last_event_time = *(double*)last_event->time;

stop_display (last_event_time);

}

/*****************************************************************************/
/* DATE: 09/30/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: start_display                                                      */
/* MODULE_NUMBER: 0.0                                                        */
/* DESCRIPTION: Writes a start display message to the dispaly file           */
/* ALGORITHM: open the diplay file                                           */
/*            write thje start display message                               */
/*            close the display file                                         */
/* PASSED VARIABLES: none                                                    */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: main()                                                   */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
void start_display ()
{
FILE *ptr_to_display_file;

if ((ptr_to_display_file = fopen ("display.c", "a")) != NULL)
{
fprintf (ptr_to_display_file, "50\n");
fclose (ptr_to_display_file);
}
else
printf ("CANNOT OPEN DISPLAY FILE IN START_DISPLAY\n");
}

/*****************************************************************************/
/* DATE: 09/30/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: stop_display                                                       */
/* MODULE_NUMBER: 1.0                                                        */
/* DESCRIPTION: Writes a stop display message to the dispaly file            */
/* ALGORITHM: open the diplay file                                           */
/*            write thje stop display message                                */
/*            close the display file                                         */
/* PASSED VARIABLES: none                                                    */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: main()                                                   */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
```

```
/**********************************************************************/
void stop_display (last_event_time)
double last_event_time;
{
FILE* ptr_to_display_file;

if ((ptr_to_display_file = fopen ("display.c", "a")) != NULL)
fprintf (ptr_to_display_file, "86 %lf\n", last_event_time);
}



/**********************************************************************/
/* DATE: 09/30/90                                                     */
/* VERSION: 0.0                                                       */
/* TITLE: compare_time                                                */
/* MODULE_NUMBER: 2.0                                                 */
/* DESCRIPTION: Used by the sim_driver to determine sorting of events */
/* ALGORITHM: subtract time1 from time2                               */
/* PASSED VARIABLES: int time1, int time2                             */
/* RETURNS: 1, or -1                                                  */
/* GLOBAL VARIABLES PASSED: none                                      */
/* GLOBAL VARIABLES CHANGED: none                                     */
/* FILES READ: none                                                   */
/* FILES WRITTEN: none                                                */
/* HARDWARE INPUT: none                                               */
/* HARDWARE OUTPUT: none                                              */
/* MODULES CALLED: none                                               */
/* CALLING MODULES: main()                                            */
/* ORDER OF: This function is of order O(1)                           */
/* AUTHOR: Rob Rizza                                                  */
/* HISTORY: none                                                      */
/**********************************************************************/
int compare_time (time1, time2)
double *time1;
double *time2;
{
if ((*time2 - *time1) < 0.0)
return -1;
else
return 1;
}



/**********************************************************************/
/* DATE: 09/30/90                                                     */
/* VERSION: 0.0                                                       */
/* TITLE: schedule_init_events                                        */
/* MODULE_NUMBER: 3.0                                                 */
/* DESCRIPTION: Schedules the first event for all objects             */
/* ALGORITHM: Pop the pointer to the first object from the master obj list */
/*            schedule the object's event                             */
/*            replace the pointer into the master obj list            */
/* PASSED VARIABLES: none                                             */
/* RETURNS: none                                                      */
/* GLOBAL VARIABLES PASSED: none                                      */
/* GLOBAL VARIABLES CHANGED: none                                     */
/* FILES READ: none                                                   */
/* FILES WRITTEN: none                                                */
/* HARDWARE INPUT: none                                               */
/* HARDWARE OUTPUT. none                                              */
/* MODULES CALLED: none                                               */
/* CALLING MODULES: main()                                            */
/* ORDER OF: This function is of order O(n) where n is the number of  */
/*           in the master object list                                */
```

```
/* AUTHOR: Rob Rizza                                                          */
/* HISTORY: none                                                              */
/****************************************************************************/
void schedule_init_events ()
{
int objects, i;
double initial_time;
double *ptr_to_initial_time = NULL;
struct object_attributes *ptr_to_object = NULL;
struct event_args *ptr_to_event_args = NULL;

objects = ll_length (master_obj_list);

for (i = 1; i <= objects; i++)
{
ptr_to_object = (struct object_attributes*)ll_pop (master_obj_list);

if (ptr_to_object->object_type <= 5)
{
if ((ptr_to_event_args = (struct event_args*)malloc
(sizeof(struct event_args))) == NULL)
printf ("CANNOT MALLOC IN SCHEDULE_INIT_EVENTS\n");

ptr_to_event_args->object1 = ptr_to_object;
ptr_to_event_args->object2 = NULL;
ptr_to_event_args->event_time = ptr_to_object->current_time;

if ((ptr_to_initial_time = (double*)malloc(sizeof(initial_time)))
 == NULL)
printf ("CANNOT MALLOC IN SCHEDULE_INIT_EVENTS\n");

*ptr_to_initial_time = ptr_to_object->current_time;

schedule_event (simulation_driver, ptr_to_initial_time,
 reached_turnpoint, ptr_to_event_args);
}
ll_insert (master_obj_list, ptr_to_object);
}
}


/****************************************************************************/
/* DATE: 09/30/90                                                           */
/* VERSION: 0.0                                                             */
/* TITLE: identify icons                                                    */
/* MODULE_NUMBER: 4.0                                                       */
/* DESCRIPTION: Send to display file the legal icons for this simulation    */
/* ALGORITHM: Open display file                                             */
/*            Send the appropriate icon identifiers                         */
/*            Close the display file                                        */
/* PASSED VARIABLES: none                                                   */
/* RETURNS: none                                                            */
/* GLOBAL VARIABLES PASSED: none                                            */
/* GLOBAL VARIABLES CHANGED: none                                           */
/* FILES READ: none                                                         */
/* FILES WRITTEN: none                                                      */
/* HARDWARE INPUT: none                                                     */
/* HARDWARE OUTPUT: none                                                    */
/* MODULES CALLED: none                                                     */
/* CALLING MODULES: main()                                                  */
/* ORDER OF: This function is of order O(1)                                 */
/* AUTHOR: Rob Rizza                                                        */
/* HISTORY: none                                                            */
/****************************************************************************/
void identify_icons ()
```

```
{
FILE *ptr_to_display_file;

if ((ptr_to_display_file = fopen ("display.c", "a")) != NULL)
{
fprintf (ptr_to_display_file, "32 1 f18\n");
fprintf (ptr_to_display_file, "32 2 migi\n");
fprintf (ptr_to_display_file, "32 3 missile\n");
fprintf (ptr_to_display_file, "32 4 tank\n");
fprintf (ptr_to_display_file, "32 5 truck\n");

fclose (ptr_to_display_file);
}
else
printf ("CANNOT OPEN DISPLAY FILE IN IDENTIFY_ICON\n");
}
```

A.3 Events Code

The following are copies of the simulation events code, events.h and events.c.

```
/************************** events.h ************************************/

void reached_turnpoint (); /* struct event_args* event_argument */

void entered_sensor_range (); /* struct event_args* event_argument */

void made_sensor_contact (); /* struct event_args* event_argument */

void collision_distance_reached (); /* struct event_args* event_argument */

void ordnance_released (); /* struct event_args *event_argument */

void ordnance_reached_target (); /* struct event_args *event_argument */
/************************************************************************/

/************************** events.c ************************************/
/************************************************************************/
/* DATE: 08/02/90                                                       */
/* VERSION: 0.0                                                         */
/* TITLE: The code for the events of this simulation                    */
/* FILENAME: events.c                                                   */
/* COORDINATOR: Rob Rizza                                               */
/* PROJECT: MS Thesis GCS-90D                                           */
/* OPERATING SYSTEM: MS-DOS                                             */
/* LANGUAGE: Micro  ft Quick-C                                          */
/* FILE PROCESSING: Link and Compile with executable file which uses this */
/*                  code.                                               */
/* CONTENTS: see the events.h code for prototypes of functions in events.c */
/* FUNCTION: Provides the simulation with legal events for the simulation */
/************************************************************************/
#include <malloc.h>
#include <stdio.h>
#include "events.h"
#include "sim_stru.h"
#include "sim_func.h"

void reached_turnpoint (event_argument)
struct event_args *event_argument;
```

```
{
printf ("Object %d has REACHED_TURNPT at %lf\n",
 event_argument->object1->object_id, event_argument->event_time);
update_position (event_argument);
sensor_check (event_argument);
}


void entered_sensor_range (event_argument)
struct event_args *event_argument;
{
printf ("Object %d has ENTERED_SENSOR_RANGE at %lf\n",
event_argument->object1->object_id, event_argument->event_time);
update_position (event_argument);
sensor_check (event_argument);
}

void made_sensor_contact (event_argument)
struct event_args *event_argument;
{
printf ("Object %d has MADE_SENSOR_CONTACT at %lf\n",
event_argument->object1->object_id, event_argument->event_time);
update_position (event_argument);
operator_evaluation (event_argument);
sensor_check (event_argument);
}

void ordnance_released (event_argument)
struct event_args *event_argument;
{
printf ("Object %d has been RELEASED as ORDNANCE at %lf\n",
event_argument->object1->object_id, event_argument->event_time);
update_position (event_argument);
sensor_check (event_argument);
}

void ordnance_reached_target (event_argument)
struct event_args *event_argument;
{
double in_x_seconds;
struct event_args *new_event_argument;

printf ("Object %d has reached it's target ORDNANCE_REACHED_TARGET at %lf\n",
event_argument->object1->object_id, event_argument->event_time);
update_position (event_argument);

in_x_seconds = event_argument->event_time - event_argument->object2->current_time;
add_new_routepoint (event_argument->object2, in_x_seconds);

if ((new_event_argument = (struct event_args*)malloc(sizeof(struct event_args)))
 == NULL)
printf ("CANNOT MALLOC NEW_EVENT_ARGUMENT IN ORDNANCE_REACHED_TARGET\n");

new_event_argument->object1 = event_argument->object2;
new_event_argument->object2 = event_argument->object1;
new_event_argument->event_time = event_argu..  ->event_time;

printf ("Object %d has ORDNANCE_REACHED_TARGET at %lf\n",
event_argument->object1->object_id, event_argument->event_time);
update_position (new_event_argument);
hit_miss (event_argument);
hit_miss (new_event_argument);
}

void collision_distance_reached (event_argument)
```

```
struct event_args *event_argument;
{
update_position (event_argument);
damage_assessment (event_argument->object1);
sensor_check (event_argument);
}
```

A.4 Functions Code

The following is a copy of the simulation functions code, sim_func.h and sim_func.c.

```
/************************************ sim_func.h ************************************/

void add_event_coords_to_route ();  /* struct event_args *event_argument */

void* add_new_routepoint (); /* struct object_attributes *object_info, double in_x_seconds */

void attack ();  /* struct event_args *event_argument */

void calc_curr_orientation (); /* struct object_attributes *object_info */

void calc_curr_velocities (); /* struct object_attributes *object_info */

double calc_time_at_next_routept (); /* struct object_attributes *object_info */

double calc_time_at_nextnext_routept (); /* struct object_attributes *object_info */

void damage_assessment ();  /* struct event_args *event_argument */

double difference_in_altitude (); /* struct event_args *event_argument */

void evade (); /* struct object_attributes *evader, struct object_attributes *evaded */

int get_sensor_range (); /* struct object_attributes *object_info */

void hit_miss ();  /* struct event_args *event_argument */

int line_of_sight (); /* struct event_args *event_argument */

int on_collision_course ();  /* struct event_args *event_argument */

int on_target_list (); /*struct object_attributes *object1, struct object_attributes *object2*/

void operator_evaluation (); /* struct event_args *event_argument */

void read_datafile (); /* char *path */

void send_fupdate (); /* struct object_attributes *object_info */

void sensor_check (); /* struct event_args *event_argument */

struct linked_list* terminate_objects (); /* struct event_args *event_argument */

void update_object_current_time ();  /* struct event_args *event_argument */

void update_position (); /*struct event_args *event_argument */
/************************************************************************************/

/************************************ sim_func.c ************************************/
```

```
/*****************************************************************************/
/* DATE: 08/02/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: Event Supporting Functions for afit_battle_sim                     */
/* FILENAME: sim_func.c                                                      */
/* COORDINATOR: Rob Rizza                                                    */
/* PROJECT: MS Thesis GCS-90D                                                */
/* OPERATING SYSTEM: MS-DOS                                                  */
/* LANGUAGE: Microsoft Quick-C                                               */
/* FILE PROCESSING: Link and Compile with executable file which uses this   */
/*                  code.                                                    */
/* CONTENTS: see the sim_func.h code for list of functions                   */
/* FUNCTION: Provides the simulation with the functions needed to run the   */
/*              simulation                                                   */
/*****************************************************************************/

#include <string.h>
#include <stdio.h>
/  * #include <process.h>   ***** comment out to run on the sun ********/
#include <math.h>
#include <malloc.h>
#include "sim_stru.h"
#include "ll.h"
#include "events.h"
#include "sim_func.h"
#include "sim_driv.h"

#define PI 3.14159
#define TRUE 1
#define FALSE 0
#define MISSLE 3

/*****************************************************************************/
/* DATE: 09/30/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: add_event_coords_to_route                                          */
/* MODULE_NUMBER: 2.0                                                        */
/* DESCRIPTION: Adds new routepoints to the objects in event_argument       */
/*              based on the time to the event                               */
/* ALGORITHM: Calculate the event time                                       */
/*            call add_new_routepoint                                        */
/* PASSED VARIABLES: event_args *event_argument                              */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: sensor_check                                             */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
void add_event_coords_to_route (event_argument)
struct event_args *event_argument;
{
double time_to_event;

if (event_argument->object1 != NULL)
{
time_to_event = event_argument->event_time - event_argument->object1->current_time;
add_new_routepoint (event_argument->object1, time_to_event);
```

```
}
if (event_argument->object2 != NULL)
{
time_to_event = event_argument->event_time - event_argument->object2->current_time;
add_new_routepoint (event_argument->object2, time_to_event);
}
}


/*****************************************************************************/
/* DATE: 08/13/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: add_new_routepoint                                                 */
/* MODULE_NUMBER: 2.1                                                        */
/* DESCRIPTION: This function is used to determine the location of the new   */
/*              turnpoint added in response to an evade request              */
/* ALGORITHM: - using the current x, y, and z velocities, add in_x_seconds   */
/*              times  each of their respective values to the current        */
/*              x, y, and z  coordinates                                     */
/* PASSED VARIABLES: object_attributes *object_info, double in_x_seconds     */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN:                                                            */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: evade, sensor_check                                      */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
void *add_new_routepoint (object_info, in_x_seconds)
struct object_attributes *object_info;
double in_x_seconds;
{
struct location_type *new_next_pt = NULL;

if ((new_next_pt = (struct location_type*)malloc(sizeof(struct location_type)))==NULL)
return NULL;

new_next_pt->x_coord = (object_info->location.x_coord + (in_x_seconds *
 object_info->velocity.x_velocity));
new_next_pt->y_coord = (object_info->location.y_coord + (in_x_seconds *
object_info->velocity.y_velocity));
new_next_pt->z_coord = (object_info->location.z_coord + (in_x_seconds *
object_info->velocity.z_velocity));

ll_insert (object_info->route_data, new_next_pt);
}


/*****************************************************************************/
/* DATE: 10/08/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: attack                                                             */
/* MODULE_NUMBER: 2.2                                                        */
/* DESCRIPTION: Creates a MISSLE object and fires it at the intended         */
/*              target                                                       */
/* ALGORITHM: Create a MISSLE object                                         */
/*            Create and load it's route points                              */
/*            Schedule an ordnance released event                            */
/* PASSED VARIABLES: event_args *event_argument                              */
```

```
/* RETURNS: none                                                              */
/* GLOBAL VARIABLES PASSED: none                                              */
/* GLOBAL VARIABLES CHANGED: none                                             */
/* FILES READ: none                                                           */
/* FILES WRITTEN: none                                                        */
/* HARDWARE INPUT: none                                                       */
/* HARDWARE OUTPUT: none                                                      */
/* MODULES CALLED: none                                                       */
/* CALLING MODULES:                                                           */
/* ORDER OF: This function is of order O(1)                                   */
/* AUTHOR: Rob Rizza                                                          */
/* HISTORY: none                                                              */
/****************************************************************************/
void attack (event_argument)
struct event_args* event_argument;
{
extern struct linked_list *master_obj_list;
extern struct driver *simulation_driver;
extern int highest_obj_id;

struct location_type *temp1, *temp2, *missle_routept1, *missle_routept2, *missle_routept3;
struct object_attributes *missle;
struct event_args *new_event_argument;
FILE *ptr_to_display_file;
double *time_ptr;

if ((missle = (struct object_attributes*)malloc(sizeof(struct object_attributes))) != NULL)
{
missle->object_type = 3;
missle->object_id = ++highest_obj_id;
missle->current_time = event_argument->event_time;
missle->location.x_coord = event_argument->object1->location.x_coord;
missle->location.y_coord = event_argument->object1->location.y_coord;
missle->location.z_coord = event_argument->object1->location.z_coord;
missle->velocity.x_velocity = 1000.0;
missle->velocity.y_velocity = 0.0;
missle->velocity.z_velocity = 0.0;
missle->orientation.yaw = 0.0;
missle->orientation.pitch = 0.0;
missle->orientation.roll = 0.0;
missle->rotation.yaw_rate = 0.0;
missle->rotation.pitch_rate = 0.0;
missle->rotation.roll_rate = 0.0;
missle->sensors = NULL;
missle->target_list = NULL;
missle->armaments = NULL;
missle->defensive_systems = NULL;

temp1 = ll_pop (event_argument->object2->route_data);

missle->route_data = ll_make (LIFO);

missle_routept3 = (struct location_type*)malloc(sizeof(struct location_type));
missle_routept3->x_coord = temp1->x_coord;
missle_routept3->y_coord = temp1->y_coord;
missle_routept3->z_coord = temp1->z_coord;

missle_routept2 = (struct location_type*)malloc(sizeof(struct location_type));
missle_routept2->x_coord = event_argument->object2->location.x_coord +
((event_argument->event_time - event_argument->object2->current_time) *
event_argument->object2->velocity.x_velocity);
missle_routept2->y_coord = event_argument->object2->location.y_coord +
((event_argument->event_time - event_argument->object2->current_time) *
event_argument->object2->velocity.y_velocity);
missle_routept2->z_coord = event_argument->object2->location.z_coord +
```

```
((event_argument->event_time - event_argument->object2->current_time) *
 event_argument->object2->velocity.z_velocity);

if (fabs (missle_routept2->x_coord - missle_routept3->x_coord) < 0.001 &&
    fabs (missle_routept2->y_coord   missle_routept3->y_coord) < 0.001 &&
    fabs (missle_routept2->z_coord - missle_routept3->z_coord) < 0.001)
{
temp2 = ll_pop (event_argument->object2->route_data);
missle_routept3->x_coord = temp2->x_coord;
missle_routept3->y_coord = temp2->y_coord;
missle_routept3->z_coord = temp2->z_coord;
ll_insert (missle->route_data, missle_routept3);
ll_insert (missle->route_data, missle_routept2);
ll_insert (event_argument->object2->route_data, temp2);
ll_insert (event_argument->object2->route_data, temp1);
}
else
{
ll_insert (event_argument->object2->route_data, temp1);
ll_insert (missle->route_data, missle_routept3);
ll_insert (missle->route_data, missle_routept2);
}
missle_routept1 = (struct location_type*)malloc(sizeof(struct location_type));
missle_routept1->x_coord = missle->location.x_coord;
missle_routept1->y_coord = missle->location.y_coord;
missle_routept1->z_coord = missle->location.z_coord;
ll_insert (missle->route_data, missle_routept1);

if ((new_event_argument = (struct event_args*)malloc(sizeof(struct event_args))) == NULL)
printf ("CANNOT MALLOC NEW_EVENT_ARGUMENT IN ATTACK");
new_event_argument->object1 = missle;
new_event_argument->object2 = event_argument->object2;
new_event_argument->event_time = event_argument->event_time;

ptr_to_display_file = fopen ("display.c", "a");
fprintf (ptr_to_display_file, "30 %d %d\n", missle->object_id, missle->object_type);
fclose (ptr_to_display_file);

ll_insert (master_obj_list, missle);

if ((time_ptr = (double*)malloc(sizeof(double))) == NULL)
printf ("CANNOT MALLOC TIME_PTR IN ATTACK");
*time_ptr = event_argument->event_time;

schedule_event (simulation_driver, time_ptr, ordnance_released, new_event_argument);
}
else
printf ("CANNOT MALLOC MISSLE IN ATTACK");
}


/*****************************************************************************/
/* DATE: 08/23/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: calc_curr_orientation                                              */
/* MODULE_NUMBER: 2.3                                                        */
/* DESCRIPTION: This function is used to determine the new orientation of a  */
/*              object based on its current and next position                */
/* ALGORITHM: - using the arctangent function calculate the angle from       */
/*              the horizontal                                               */
/* PASSED VARIABLES: object_info                                             */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
```

```
/* FILES WRITTEN:                                                          */
/* HARDWARE INPUT: none                                                    */
/* HARDWARE OUTPUT: none                                                   */
/* MODULES CALLED: none                                                    */
/* CALLING MODULES: evade                                                  */
/* ORDER OF: This function is of order O(1)                                */
/* AUTHOR: Rob Rizza                                                       */
/* HISTORY: none                                                           */
/*-------------------------------------------------------------------------*/
void calc_curr_orientation (object_info)
struct object_attributes *object_info;
{
double delta_x, delta_y, delta_z, distance, angle, pitch;
struct location_type *next_route_point = NULL;

if (ll_isempty (object_info->route_data) != TRUE)
{
next_route_point = (struct location_type*)ll_pop (object_info->route_data);
delta_x = next_route_point->x_coord - object_info->location.x_coord;
delta_y = next_route_point->y_coord - object_info->location.y_coord;
delta_z = next_route_point->z_coord - object_info->location.z_coord;

angle = atan2 (delta_y, delta_x) * 360 / (2 * PI);

if (angle < 0.0)
angle = 360 + angle;

if (angle >= 0.0 && angle <= 90.0)
object_info->orientation.yaw = 90.0 - angle;
else if (angle > 90.0 && angle <= 180.0)
object_info->orientation.yaw = 360.0 - (angle - 90.0);
else if (angle > 180.0 && angle <= 270.0)
object_info->orientation.yaw = 270.0 - (angle - 180.0);
else
object_info->orientation.yaw = 180 - (angle - 270.0);

pitch = atan2 (delta_z, (distance - sqrt ((delta_x*delta_x)+(delta_y*delta_y)))) *
 360 / (2 * PI);

object_info->orientation.pitch = pitch;
object_info->orientation.roll = object_info->orientation.roll;

ll_insert (object_info->route_data, next_route_point);
}
else
{
object_info->orientation.pitch = 0.0;
object_info->orientation.roll = 0.0;
}
}


/*-------------------------------------------------------------------------*/
/* DATE: 08/24/90                                                          */
/* VERSION: 0.0                                                            */
/* TITLE: calc_curr_velocities                                             */
/* MODULE_NUMBER: 2.4                                                      */
/* DESCRIPTION: This function is used to determine the new velocity vectors */
/*              of a vehicle based on its next route point                 */
/* ALGORITHM: - using the arctangent function calculate the angle from     */
/*              the horizontal                                             */
/*            - then use the cosine and sine functions multiplied by the   */
/*              total velocity vectors to find the new velocity vectors    */
/* PASSED VARIABLES: object_info                                           */
/* RETURNS: none                                                           */
```

```
/* GLOBAL VARIABLES PASSED: none                                          */
/* GLOBAL VARIABLES CHANGED: none                                         */
/* FILES READ: none                                                       */
/* FILES WRITTEN:                                                         */
/* HARDWARE INPUT: none                                                   */
/* HARDWARE OUTPUT: none                                                  */
/* MODULES CALLED: none                                                   */
/* CALLING MODULES: evade, update_position                                */
/* ORDER OF: This function is of order O(1)                               */
/* AUTHOR: Rob Rizza                                                      */
/* HISTORY: none                                                          */
/************************************************************************/

void calc_curr_velocities (object_info)
struct object_attributes *object_info;
{
struct location_type *next_route_point = NULL;
double delta_x, delta_y, delta_z, slope_angle, horizontal_vel_vector,
       time_to_next_route_point, distance_to_next_route_point;

if (ll_isempty (object_info->route_data) != TRUE)
{
next_route_point = (struct location_type*)ll_pop (object_info->route_data);
horizontal_vel_vector = sqrt
((object_info->velocity.x_velocity * object_info->velocity.x_velocity) +
 (object_info->velocity.y_velocity * object_info->velocity.y_velocity));

delta_x = next_route_point->x_coord - object_info->location.x_coord;
delta_y = next_route_point->y_coord - object_info->location.y_coord;
delta_z = next_route_point->z_coord - object_info->location.z_coord;

slope_angle = atan2 (delta_y, delta_x);
distance_to_next_route_point = sqrt ((delta_x * delta_x) + (delta_y * delta_y));
time_to_next_route_point = distance_to_next_route_point / horizontal_vel_vector;

object_info->velocity.x_velocity = horizontal_vel_vector * cos(slope_angle);
object_info->velocity.y_velocity = horizontal_vel_vector * sin(slope_angle);
object_info->velocity.z_velocity = delta_z / time_to_next_route_point;

ll_insert (object_info->route_data, next_route_point);
}
else
{
object_info->velocity.x_velocity = 0.0;
object_info->velocity.y_velocity = 0.0;
object_info->velocity.z_velocity = 0.0;
}
}


/************************************************************************/
/* DATE: 09/06/90                                                       */
/* VERSION: 0.0                                                         */
/* TITLE: calc_time_at_next_routept                                     */
/* MODULE_NUMBER: 2.4                                                   */
/* DESCRIPTION: This function is used to determine the time at the next */
/*              routepoint based on the distance travelled and the      */
/*              current velocity                                        */
/* ALGORITHM: - pop the next routepoint off the route data queue        */
/*            - using the standard distance formula between 2 points    */
/*                 find the distance travelled                          */
/*            - calculate the total velocity vector                     */
/*            - time at next routepoint = distance travelled /          */
/*                                        total velocity vector + curr_time */
/* PASSED VARIABLES: object_info                                        */
```

```
/* RETURNS: double time_at_next_routept                                    */
/* GLOBAL VARIABLES PASSED: none                                           */
/* GLOBAL VARIABLES CHANGED: none                                          */
/* FILES READ: none                                                        */
/* FILES WRITTEN:                                                          */
/* HARDWARE INPUT: none                                                    */
/* HARDWARE OUTPUT: none                                                   */
/* MODULES CALLED: none                                                    */
/* CALLING MODULES: update_position                                        */
/* ORDER OF: This function is of order O(1)                                */
/* AUTHOR: Rob Rizza                                                       */
/* HISTORY: none                                                           */
/*-------------------------------------------------------------------------*/
double calc_time_at_next_routept (object_info)
struct object_attributes *object_info;
{
double delta_x, delta_y, delta_z, distance_traveled, time_at_next_routept, total_vel_vector;
struct location_type *next_routept = NULL;
int event;

time_at_next_routept = object_info->current_time;
if (ll_isempty (object_info->route_data) != TRUE)
{
next_routept = (struct location_type*)ll_pop (object_info->route_data);
delta_x = object_info->location.x_coord - next_routept->x_coord;
delta_y = object_info->location.y_coord - next_routept->y_coord;
delta_z = object_info->location.z_coord - next_routept->z_coord;

ll_insert (object_info->route_data, next_routept);

distance_traveled = sqrt ((delta_x*delta_x) + (delta_y*delta_y) + (delta_z*delta_z));

total_vel_vector = sqrt ((object_info->velocity.x_velocity *
 object_info->velocity.x_velocity) +
      (object_info->velocity.y_velocity * object_info->velocity.y_velocity) +
      (object_info->velocity.z_velocity * object_info->velocity.z_velocity));

if (total_vel_vector != 0.0)
time_at_next_routept = object_info->current_time +
 distance_traveled / total_vel_vector;
else
time_at_next_routept = object_info->current_time;

}
return time_at_next_routept;
}


/*-------------------------------------------------------------------------*/
/* DATE: 09/06/90                                                          */
/* VERSION: 0.0                                                            */
/* TITLE: calc_time_at_nextnext_routept                                    */
/* MODULE_NUMBER: 2.6                                                      */
/* DESCRIPTION: This function is used to determine the time at the next    */
/*              routepoint based on the distance travelled and the         */
/*              current velocity                                           */
/* ALGORITHM: - pop the next routepoint off the route data queue           */
/*            - using the standard distance formula between 2 points       */
/*                  find the distance travelled                            */
/*            - calculate the total velocity vector                        */
/*            - time at next routepoint = distance travelled /             */
/*                                        total velocity vector            */
/* PASSED VARIABLES: object_info                                           */
/* RETURNS: double time_at_next_routept                                    */
/* GLOBAL VARIABLES PASSED: none                                           */
```

```
/* GLOBAL VARIABLES CHANGED: none                                          */
/* FILES READ: none                                                        */
/* FILES WRITTEN:                                                          */
/* HARDWARE INPUT: none                                                    */
/* HARDWARE OUTPUT: none                                                   */
/* MODULES CALLED: none                                                    */
/* CALLING MODULES: update_position                                        */
/* ORDER OF: This function is of order O(1)                                */
/* AUTHOR: Rob Rizza                                                       */
/* HISTORY: none                                                           */
/*************************************************************************/
double calc_time_at_nextnext_routept (object_info)
struct object_attributes *object_info;
{

double delta_x, delta_y, delta_z, distance_traveled, time_at_next_routept, total_vel_vector;
struct location_type *next_routept = NULL, *nextnext_routept = NULL;
int event;

time_at_next_routept = object_info->current_time;

if (ll_isempty (object_info->route_data) != TRUE)
{
next_routept = (struct location_type*)ll_pop (object_info->route_data);
delta_x = object_info->location.x_coord - next_routept->x_coord;
delta_y = object_info->location.y_coord - next_routept->y_coord;
delta_z = object_info->location.z_coord - next_routept->z_coord;

distance_traveled = sqrt ((delta_x*delta_x) + (delta_y*delta_y) + (delta_z*delta_z));

total_vel_vector = sqrt ((object_info->velocity.x_velocity *
object_info->velocity.x_velocity) +
      (object_info->velocity.y_velocity * object_info->velocity.y_velocity) +
      (object_info->velocity.z_velocity * object_info->velocity.z_velocity));

if (total_vel_vector != 0.0)
time_at_next_routept = object_info->current_time +
 distance_traveled / total_vel_vector;
else
time_at_next_routept = object_info->current_time;

}
if (ll_isempty (object_info->route_data) != TRUE)
{
nextnext_routept = (struct location_type*)ll_pop (object_info->route_data);
delta_x = next_routept->x_coord - nextnext_routept->x_coord;
delta_y = next_routept->y_coord - nextnext_routept->y_coord;
delta_z = next_routept->z_coord - nextnext_routept->z_coord;

distance_traveled = sqrt ((delta_x*delta_x) + (delta_y*delta_y) + (delta_z*delta_z));

total_vel_vector = sqrt ((object_info->velocity.x_velocity *
 object_info->velocity.x_velocity) +
      (object_info->velocity.y_velocity * object_info->velocity.y_velocity) +
      (object_info->velocity.z_velocity * object_info->velocity.z_velocity));

ll_insert (object_info->route_data, nextnext_routept);
ll_insert (object_info->route_data, next_routept);

if (total_vel_vector != 0.0)
time_at_next_routept = time_at_next_routept + distance_traveled /
 total_vel_vector;
else
time_at_next_routept = time_at_next_routept;
```

```
return time_at_next_routept;
}

if (next_routept != NULL && nextnext_routept == NULL)
ll_insert (object_info->route_data, next_routept);

return time_at_next_routept;
}


/*****************************************************************************/
/* DATE: 09/20/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: damage_assessment                                                  */
/* MODULE_NUMBER: 2.7                                                        */
/* DESCRIPTION: Determine extent of damage                                   */
/*              Schedule appropriate event                                   */
/* ALGORITHM: TBD                                                            */
/*                                                                           */
/*                                                                           */
/* PASSED VARIABLES: event_args *event_argument                              */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: ordnance_reached_target                                  */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
void damage_assessment (event_argument)
struct event_args *event_argument;
{
terminate_objects (event_argument);
}


/*****************************************************************************/
/* DATE: 09/20/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: difference_in_altitude                                             */
/* MODULE_NUMBER: 2.8                                                        */
/* DESCRIPTION: Determines the difference in altitude of two objects         */
/* ALGORITHM: Determine the current altitude of the objects                  */
/*            Return their difference                                        */
/* PASSED VARIABLES: event_args *event_argument                              */
/* RETURNS: double difference or 0.0                                         */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: sensor_check, collision distance reached                 */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
double difference_in_altitude (event_argument)
```

```
struct event_args *event_argument;
{

double difference, curr_time;

curr_time = event_argument->object1->current_time;

if ((difference = fabs (event_argument->object1->location.x_coord -
    (event_argument->object2->location.x_coord +
    ((curr_time - event_argument->object2->current_time) *
    event_argument->object2->velocity.x_velocity)))) <= 5.0)
return 0.0;
else
return difference;
}


/*****************************************************************************/
/* DATE: 08/13/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: evade                                                              */
/* MODULE_NUMBER: 2.9                                                        */
/* DESCRIPTION: This function is used to reorient and change the velocity    */
/*              vectors of a vehicle in response to it turning away from a   */
/*              threat                                                       */
/* ALGORITHM: - calculate or determine the threat vehicle's path             */
/*            - adjust your path to be 90 degrees from the threat path       */
/*              moving away from the threat path                             */
/* PASSED VARIABLES: evader, and evaded                                      */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN:                                                            */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: send_(f)update, add_new_routepoint                        */
/* CALLING MODULES: operator_evaluation                                      */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
void evade (evader, evaded)
struct object_attributes *evader;
struct object_attributes *evaded;
{
int is_routept_good = TRUE;
double evaded_slope, evader_slope, evaded_y_intercept, evader_y_intercept;
double x_direction_indicator, y_direction_indicator, common_x_pt, common_y_pt;
double slope_angle, relative_position, total_vel_vector, x1_temp, y1_temp;
double x2_temp, y2_temp, delta_x, delta_y, dist1, dist2;

struct location_type *next_routept = NULL;

total_vel_vector = sqrt ((evader->velocity.x_velocity * evader->velocity.x_velocity) +
 (evader->velocity.y_velocity * evader->velocity.y_velocity));

if (evaded->velocity.x_velocity == 0.0 && evaded->velocity.y_velocity != 0.0)
{
relative_position = evader->location.x_coord - evaded->location.x_coord;

if (relative_position < 0.0)
{
evader->velocity.x_velocity = total_vel_vector * -1.0;
evader->velocity.y_velocity = 0.0;
```

```
evader->velocity.z_velocity = 0.0;

evader->orientation.yaw = 270.0;
evader->orientation.pitch = 0.0;
evader->orientation.roll = 0.0;
}
else
{
evader->velocity.x_velocity = total_vel_vector;
evader->velocity.y_velocity = 0.0;
evader->velocity.z_velocity = 0.0;

evader->orientation.yaw = 90.0;
evader->orientation.pitch = 0.0;
evader->orientation.roll = 0.0;
}
add_new_routepoint (evader, 60.0);
}

else if (evaded->velocity.y_velocity == 0.0 && evaded->velocity.x_velocity != 0.0)
{
relative_position = evader->location.y_coord - evaded->location.y_coord;

if (relative_position < 0.0)
{
evader->velocity.x_velocity = 0.0;
evader->velocity.y_velocity = total_vel_vector * -1.0;
evader->velocity.z_velocity = 0.0;

evader->orientation.yaw = 180.0;
evader->orientation.pitch = 0.0;
evader->orientation.roll = 0.0;
}
else
{
evader->velocity.x_velocity = 0.0;
evader->velocity.y_velocity = total_vel_vector;
evader->velocity.z_velocity = 0.0;

evader->orientation.yaw = 0.0;
evader->orientation.pitch = 0.0;
evader->orientation.roll = 0.0;
}
add_new_routepoint (evader, 60.0);
}

else if (evaded->velocity.x_velocity == 0.0 && evaded->velocity.y_velocity == 0.0)
{
delta_x = evader->location.x_coord - evaded->location.x_coord;
delta_y = evader->location.y_coord - evaded->location.y_coord;

slope_angle = atan2 ((delta_x * -1), delta_y);

x1_temp = evader->location.x_coord + (10 * cos (slope_angle));
y1_temp = evader->location.y_coord + (10 * sin (slope_angle));

x2_temp = evader->location.x_coord - (10 * cos (slope_angle));
y2_temp = evader->location.y_coord - (10 * sin (slope_angle));

next_routept = ll_pop (evader->route_data);

if (sqrt ((next_routept->x_coord - evaded->location.x_coord) *
  (next_routept->x_coord - evaded->location.x_coord) +
  (next_routept->y_coord - evaded->location.y_coord) *
  (next_routept->y_coord - evaded->location.y_coord)) <=
```

```
   (double)get_sensor_range (evader))
{
is_routept_good = FALSE;
if (ll_isempty (evader->route_data) != TRUE)
{
next_routept = ll_pop (evader->route_data);
is_routept_good = TRUE;
}
}

x1_temp = x1_temp - next_routept->x_coord;
y1_temp = y1_temp - next_routept->y_coord;

x2_temp = x2_temp - next_routept->x_coord;
y2_temp = y2_temp - next_routept->y_coord;

dist1 = sqrt((x1_temp * x1_temp) + (y1_temp * y1_temp));
dist2 = sqrt((x2_temp * x2_temp) + (y2_temp * y2_temp));

if (dist1 <= dist2)
{
evader->velocity.x_velocity = total_vel_vector * cos (slope_angle);
evader->velocity.y_velocity = total_vel_vector * sin (slope_angle);
evader->velocity.z_velocity = 0.0;
}
else
{
evader->velocity.x_velocity = total_vel_vector * cos (slope_angle) * -1;
evader->velocity.y_velocity = total_vel_vector * sin (slope_angle) * -1;
evader->velocity.z_velocity = 0.0;
}

if (is_routept_good == TRUE)
ll_insert (evader->route_data, next_routept);
add_new_routepoint (evader, 60.0);
calc_curr_orientation (evader);
}

else
{
evaded_slope = evaded->velocity.y_velocity / evaded->velocity.x_velocity;
evaded_y_intercept = evaded->location.y_coord - (evaded_slope *
evaded->location.x_coord);

evader_slope = -1 / evaded_slope;
evader_y_intercept = evader->location.y_coord - (evader_slope *
evader->location.x_coord);

common_x_pt = (evaded_y_intercept - evader_y_intercept) / (-1 *
(evaded_slope - evader_slope));
common_y_pt = evaded_slope * common_x_pt + evaded_y_intercept;

x_direction_indicator = evader->location.x_coord - common_x_pt;
y_direction_indicator = evader->location.y_coord - common_y_pt;

slope_angle = atan (evader_slope);

evader->velocity.x_velocity = fabs (total_vel_vector * cos (slope_angle));
evader->velocity.y_velocity = fabs (total_vel_vector * sin (slope_angle));
evader->velocity.z_velocity = 0.0;

if (x_direction_indicator < 0.0)
      evader->velocity.x_velocity = -1 * evader->velocity.x_velocity;

if (y_direction_indicator < 0.0)
```

```
      evader->velocity.y_velocity = -1 * evader->velocity.y_velocity;

add_new_routepoint (evader, 60.0);
calc_curr_orientation (evader);
}
send_fupdate (evader);
}



/*****************************************************************************/
/* DATE: 09/20/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: get_sensor_range                                                   */
/* MODULE_NUMBER: 2.10                                                       */
/* DESCRIPTION: This function is used by sensor_check to determine the       */
/*              range of the sensor being used                               */
/* ALGORITHM: For as many items that there are in the sensor list            */
/*              - check the sensor range, save the largest range found       */
/*              - return the range found                                     */
/* PASSED VARIABLES: struct object_attributes* object_info                   */
/* RETURNS: range                                                            */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: sensor_check                                             */
/* ORDER OF: This function is of order O(n) where n is the number of sensors*/
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
int get_sensor_range (object_info)
struct object_attributes *object_info;
{
int i, length, range = 833, temp_range = 0;   /* default sensor range, approx 1/2 mile */
struct sensors *sensor = NULL;

if (object_info->sensors != NULL)
{
length = ll_length (object_info->sensors);

for (i = 1; i <= length; i++)
{
sensor = (struct sensors*)ll_pop (object_info->sensors);
if (sensor->range > range)
range = sensor->range;
ll_insert (object_info->sensors, sensor);
}
}
else
if (object_info->object_type == MISSLE)
range = 0;

return range;
}


/*****************************************************************************/
/* DATE: 09/20/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: hit_miss                                                           */
/* MODULE_NUMBER: 2.11                                                       */
```

```
/* DESCRIPTION: Determines if a hit or miss takes place                    */
/* ALGORITHM: If the objects come within a specified distance              */
/*                  schedule a damage_assessment                           */
/*            Else                                                         */
/*                  call sensor_check                                      */
/* PASSED VARIABLES: event_args *event_argument                            */
/* RETURNS: none                                                           */
/* GLOBAL VARIABLES PASSED: none                                           */
/* GLOBAL VARIABLES CHANGED: none                                          */
/* FILES READ: none                                                        */
/* FILES WRITTEN: none                                                     */
/* HARDWARE INPUT: none                                                    */
/* HARDWARE OUTPUT: none                                                   */
/* MODULES CALLED: none                                                    */
/* CALLING MODULES: ordnace_reached_target                                 */
/* ORDER OF: This function is of order O(1)                                */
/* AUTHOR: Rob Rizza                                                       */
/* HISTORY: none                                                           */
/*-------------------------------------------------------------------------*/
void hit_miss (event_argument)
struct event_args *event_argument;
{
      if (fabs (event_argument->object1->location.x_coord -
event_argument->object2->location.x_coord) <= 10.0 &&
  fabs (event_argument->object1->location.y_coord -
event_argument->object2->location.y_coord) <= 10.0 &&
  fabs (event_argument->object1->location.z_coord -
 event_argument->object2->location.z_coord) <= 10.0)
damage_assessment (event_argument);
      else
sensor_check (event_argument);
}



/*-------------------------------------------------------------------------*/
/* DATE: 09/20/90                                                          */
/* VERSION: 0.0                                                            */
/* TITLE: line_of_sight                                                    */
/* MODULE_NUMBER: 2.12                                                     */
/* DESCRIPTION: This function is used by sensor_check to determine if an   */
/*              unobstructed line of sight exists between two objects      */
/* ALGORITHM: TBD                                                          */
/*                                                                         */
/*                                                                         */
/* PASSED VARIABLES: struct event_args* event_argument                     */
/* RETURNS: 0, 1                                                           */
/* GLOBAL VARIABLES PASSED: none                                           */
/* GLOBAL VARIABLES CHANGED: none                                          */
/* FILES READ: none                                                        */
/* FILES WRITTEN: none                                                     */
/* HARDWARE INPUT: none                                                    */
/* HARDWARE OUTPUT: none                                                   */
/* MODULES CALLED: none                                                    */
/* CALLING MODULES: sensor_check                                           */
/* ORDER OF: This function is of order O(1)                                */
/* AUTHOR: Rob Rizza                                                       */
/* HISTORY: none                                                           */
/*-------------------------------------------------------------------------*/
int line_of_sight (event_argument)
struct event_args *event_argument;
{
return 1;
}
```

```
/*****************************************************************************/
/* DATE: 09/30/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: on_collision_course                                                */
/* MODULE_NUMBER: 2.13                                                       */
/* DESCRIPTION:  Determines whether two objects will occupy the same         */
/*               location at the same time                                   */
/* ALGORITHM: Determine if the objects will occupy the same position at the  */
/*            same time                                                      */
/*            Return TRUE if true                                            */
/*            Else return FALSE                                              */
/* PASSED VARIABLES: event_args *event_argument                              */
/* RETURNS: TRUE or FALSE                                                    */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES:                                                          */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
int on_collision_course (event_argument)
struct event_args *event_argument;
{
double diff_in_curr_x_coords, diff_in_curr_y_coords, diff_in_curr_x_vels,
       diff_in_curr_y_vels, a, b, c, term_under_radical, time_at_next_routept1,
       time_at_next_routept2, sensor_contact_time, curr_time;

curr_time = event_argument->object1->current_time;

diff_in_curr_x_coords = event_argument->object1->location.x_coord -
(event_argument->object2->location.x_coord +
((curr_time - event_argument->object2->current_time)
 * event_argument->object2->velocity.x_velocity));
diff_in_curr_y_coords = event_argument->object1->location.y_coord -
 (event_argument->object2->location.y_coord +
((curr_time - event_argument->object2->current_time)
* event_argument->object2->velocity.y_velocity));

diff_in_curr_x_vels = event_argument->object1->velocity.x_velocity -
event_argument->object2->velocity.x_velocity;
diff_in_curr_y_vels = event_argument->object1->velocity.y_velocity -
event_argument->object2->velocity.y_velocity;


       /************  QUADRATIC EQUATION IN (t): t = (-b +- sqrt (bb - 4ac))/2a  **********/

a = (diff_in_curr_x_vels * diff_in_curr_x_vels) +
    (diff_in_curr_y_vels * diff_in_curr_y_vels);
b = (2.0 * diff_in_curr_x_coords * diff_in_curr_x_vels) +
    (2.0 * diff_in_curr_y_coords * diff_in_curr_y_vels);
c = (diff_in_curr_x_coords * diff_in_curr_x_coords) +
    (diff_in_curr_y_coords * diff_in_curr_y_coords);

term_under_radical = (b * b) - (4.0 * a* c);

if (fabs (term_under_radical) < 0.0001)
term_under_radical = 0.0;
```

```
if (term_under_radical == 0.0 && difference_in_altitude(event_argument) == 0.0)
{
time_at_next_routept1 = calc_time_at_next_routept (event_argument->object1);
time_at_next_routept2 = calc_time_at_next_routept (event_argument->object2);

sensor_contact_time = (-1 * b - sqrt (term_under_radical)) /
      (2.0 * a) + event_argument->object1->current_time;

if (sensor_contact_time <= time_at_next_routept1 &&
    sensor_contact_time <= time_at_next_routept2)
return TRUE;
}

return FALSE;
}


/*****************************************************************************/
/* DATE: 09/30/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: on_target_list                                                     */
/* MODULE NUMBER: 2.14                                                       */
/* DESCRIPTION: Determines whether an object is on another object's target   */
/*              list                                                         */
/* ALGORITHM: Search an object's target list for the other object            */
/*            Return TRUE if found                                           */
/*            Else return FALSE                                              */
/* PASSED VARIABLES: object_attributes *object1, object_atrributes *object2  */
/* RETURNS: int TRUE or FALSE                                                */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES:                                                          */
/* ORDER OF: This function is of order O(n) where n is the number of         */
/*            targets in the objects target list                             */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
int on_target_list (object1, object2)
struct object_attributes *object1;
struct object_attributes *object2;
{
int num_targets, i, return_value = FALSE;
struct targets *target;

if (object1->target_list != NULL)
{
num_targets = ll_length (object1->target_list);

for (i = 1; i <= num_targets; i++)
{
target = ll_pop (object1->target_list);

if (target->target_type == object2->object_type)
return_value = TRUE;

ll_insert (object1->target_list, target);
}
}
return return_value;
```

```
}


/*****************************************************************************/
/* DATE: 09/30/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: operator_evaluation                                                */
/* MODULE_NUMBER: 2.15                                                       */
/* DESCRIPTION: Determines the next course of action for an object which     */
/*              has sensed another object                                    */
/* ALGORITHM: see case statement below                                       */
/* PASSED VARIABLES: event_args *event_argument                              */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES:                                                          */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
void operator_evaluation (event_argument)
struct event_args *event_argument;
{
struct object_attributes *observer = NULL;
struct object_attributes *observed = NULL;
int event, sensor_range_observer, sensor_range_observed;

if (event_argument->object1 != NULL)
observer = event_argument->object1;

if (event_argument->object2 != NULL)
observed = event_argument->object2;

sensor_range_observer = get_sensor_range (observer);
sensor_range_observed = get_sensor_range (observed);

if (observed->object_type == MISSLE)
event = 0x00;      /* do nothing */

else if ((observer->object_loyalty == observed->object_loyalty) &&
   (on_collision_course (event_argument) == FALSE))
event = 0x00;      /* do nothing */

else if ((observer->object_loyalty == observed->object_loyalty) &&
((on_collision_course (event_argument)) == TRUE))
event = 0x01;       /* evade */

else if ((observer->object_loyalty != observed->object_loyalty) &&
(sensor_range_observer > sensor_range_observed) &&
(on_target_list (observer, observed) == FALSE))
event = 0x01;      /* evade */

else if (observer->object_loyalty != observed->object_loyalty &&
(on_target_list (observer, observed) == TRUE ||
sensor_range_observer <= sensor_range_observed))
event = 0x10;      /* attack */
switch (event)
{
case 0x00:
```

```
break;

case 0x01:
{
/* put in code to make sure next routept is
   not within sensor range of the stationary
   object being avoided */

evade (observer, observed);

break;
}
case 0x10:
attack (event_argument);
}
}



/*****************************************************************************/
/* DATE: 08/31/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: read_datfile                                                       */
/* MODULE_NUMBER: 2.16                                                       */
/* DESCRIPTION: This function is used to read the scenario data from file    */
/* ALGORITHM: - while the pointer has not reached the end of file            */
/*                - read in a line                                           */
/*                - assign the data to it appropriate field                  */
/*                - write the icon identifying info to the display file      */
/* PASSED VARIABLES: path                                                    */
/* RETURNS: struct linked_list* master_obj_list                              */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN:                                                            */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: main                                                     */
/* ORDER OF: This function is of order O(n) where n is the number of lines   */
/*                                        in the file being read             */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
void read_datafile (path)
char *path;
{
FILE *ptr_to_datafile, *ptr_to_display_file;

extern struct linked_list *master_obj_list;
extern int highest_obj_id;

struct object_attributes *object = NULL;
struct location_type *routept = NULL;
struct sensors *sensor = NULL;
struct targets *target = NULL;
struct armaments *armament = NULL;
struct defensive_systems *defensive_system = NULL;

int i, fields, num_fields, line_num = 0, num_routepts = 0, num_targets = 0;
int num_sensors = 0, num_armaments = 0, num_defensive_systems = 0, object_type;

char line[400], *ptr_to_line = NULL;
char *temp_ptr = NULL;
```

```
if ((ptr_to_datafile = fopen(path, "r")) != NULL)
{
while (!feof (ptr_to_datafile))
{
if (fgets (line, 400, ptr_to_datafile) == NULL)
break;

if ((object = (struct object_attributes*)malloc
(sizeof(struct object_attributes))) != NULL)
{
++line_num;

object->object_type = atoi (strtok (line, " "));
object->object_id = atoi (strtok (NULL, " "));
object->object_loyalty = atoi (strtok (NULL, " " ));
object->current_time = atof (strtok (NULL, " "));
object->fuel_status = atoi (strtok (NULL, " "));
object->condition = atoi (strtok (NULL, " "));
object->vulnerability = atoi (strtok (NULL, " "));
object->location.x_coord = atof (strtok (NULL, " "));
object->location.y_coord = atof (strtok (NULL, " "));
object->location.z_coord = atof (strtok (NULL, " "));
object->velocity.x_velocity = atof (strtok (NULL, " "));
object->velocity.y_velocity = atof (strtok (NULL, " "));
object->velocity.z_velocity = atof (strtok (NULL, " "));
object->rotation.yaw_rate = atof (strtok (NULL, " "));
object->rotation.pitch_rate = atof (strtok (NULL, " "));
object->rotation.roll_rate = atof (strtok (NULL, " "));
object->operator.experience = atoi (strtok (NULL, " "));
object->operator.threat_knowledge = atoi (strtok (NULL, " "));
object->performance.min_turn_radius = atoi (strtok (NULL, " "));
object->performance.max_speed = atoi (strtok (NULL, " "));
object->performance.ave_fuel_cons_rate = atoi (strtok (NULL, " "));
object->performance.max_climb_rate = atoi (strtok (NULL, " "));

num_routepts = atoi (strtok (NULL, " "));

object->route_data = ll_make (LIFO);

for (i = 1; i <= num_routepts; i++)
{
if ((routept = (struct location_type*)malloc
(sizeof(struct location type))) != NULL)
{
routept->x_coord = atof (strtok (NULL, " "));
routept->y_coord = atof (strtok (NULL, " "));
routept->z_coord = atof (strtok (NULL, " "));

ll_insert (object->route_data, routept);
}
else
printf ("CANNOT READ IN ROUTEPOINTS, NOT ENOUGH MEMORY\n");
}

if ((num_sensors = atoi (strtok (NULL, " "))) > 0)
{
object->sensors = ll_make (FIFO);
for (i = 1; i <= num_sensors; i++)
{

if ((sensor = (struct sensors*)
malloc(sizeof(struct sensors))) != NULL)
{
sensor->type = atoi (strtok (NULL, " "));
sensor->range = atoi (strtok (NULL, " "));
```

```
sensor->resolution = atoi (strtok (NULL, " "));

ll_insert (object->sensors, sensor);
}
else
printf ("CANNOT READ IN SENSORS, NOT ENOUGH MEMORY\n");
}
 }
 else
object->sensors = NULL;

if ((num_armaments = atoi (strtok (NULL, " "))) > 0)
{
object->armaments = ll_make (FIFO);
for (i = 1; i <= num_armaments; i++)
{

if ((armament = (struct armaments*)
malloc(sizeof(struct armaments))) != NULL)
{
armament->type = atoi (strtok (NULL, " "));
armament->range = atoi (strtok (NULL, " "));
armament->lethality = atoi (strtok (NULL, " "));
armament->accuracy = atoi (strtok (NULL, " "));
armament->speed = atoi (strtok (NULL, " "));
armament->count = atoi (strtok (NULL, " "));

ll_insert (object->armaments, armament);
}
else
printf ("CANNOT READ IN ARMAMENTS, NOT ENOUGH MEMORY\n");
}
}
else
object->armaments = NULL;

if ((num_targets = atoi (strtok (NULL, " "))) > 0)
{
object->target_list = ll_make (FIFO);
for (i = 1; i <= num_targets; i++)
{

if ((target = (struct targets*)malloc(sizeof(struct targets)))
 != NULL)
{
target->target_type = atoi (strtok (NULL, " "));
target->target_location.x_coord = atof (strtok
(NULL, " "));
target->target_location.y_coord = atof (strtok
(NULL, " "));
target->target_location.z_coord = atof (strtok
(NULL, " "));

ll_insert (object->target_list, target);
}
else
printf ("CANNOT READ IN TARGETS, NOT ENOUGH MEMORY\n");
}
}
else
object->target_list = NULL;

temp_ptr = strtok (NULL, " ");

if (temp_ptr == NULL)
```

```
printf ("BOGUS DATA IN LINE %d OF INPUT DATAFILE\n", line_num);

if ((num_defensive_systems = atoi (temp_ptr)) > 0)
{
object->defensive_systems = ll_make (FIFO);
for (i = 1; i <= num_defensive_systems; i++)
{

if ((defensive_system = (struct defensive_systems*)
malloc(sizeof(struct defensive_systems))) != NULL)
{
defensive_system->type = atoi (strtok (NULL, " "));
defensive_system->range = atoi (strtok (NULL, " "));
temp_ptr = strtok (NULL, " ");

if (temp_ptr == NULL)
printf ("BOGUS DATA IN LINE %d OF INPUT
 DATAFILE\n", line_num);

defensive_system->effectiveness = atoi (temp_ptr),

ll_insert (object->defensive_systems, defensive_system);
}
else
printf ("CANNOT READ IN DEFENSIVE SYSTEMS,
 NOT ENOUGH MEMORY\n");
}
}
else
object->defensive_systems = NULL;

if ((ptr_to_display_file = fopen ("display.c", "a")) != NULL)
{
fprintf (ptr_to_display_file, "30 %d %d\n",
 object->object_id, object->object_type);
fclose (ptr_to_display_file);

ll_insert (master_obj_list, object);
}
else
printf ("CANNOT OPEN DISPLAY FILE IN READ_DATAFILE\n");
}
else
printf ("CANNOT READ IN VEHICLE ATTRIBUTES, NOT ENOUGH MEMORY\n");
}
}
else
printf ("CANNOUT OPEN VEHICLE FILE FOR READING\n");

highest_obj_id = object->object_id;
fclose (ptr_to_datafile);

}



/*****************************************************************************/
/* DATE: 08/02/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: send_fupdate                                                       */
/* MODULE_NUMBER: 2.17                                                       */
/* DESCRIPTION: This function is used to send position updates to a file     */
/*              for later access by the generic display                      */
/* ALGORITHM: open a file to store the information                           */
/*            extract and read the required data into the datafile           */
```

```
/*            close the datafile once the reading is complete                */
/* PASSED VARIABLES: object_info                                             */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: datafile                                                   */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: reach_turnpoint, reached_target, reached_destination,    */
/*                  reached_sensor_range                                     */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/***************************************************************************/
void send_fupdate (object_info)
struct object_attributes *object_info;
{
FILE *ptr_to_display_file;

if ((ptr_to_display_file = fopen("display.c", "a")) != NULL)
{
fprintf (ptr_to_display_file, "31 %d %lf %.2lf %.2lf %.2lf %lf %lf %lf %lf
 %lf %lf %lf %lf %lf\n", object_info->object_id,
 object_info->current_time, object_info->location.x_coord,
 object_info->location.y_coord, object_info->location.z_coord,
 object_info->velocity.x_velocity, object_info->velocity.y_velocity,
 object_info->velocity.z_velocity, object_info->orientation.yaw,
 object_info->orientation.pitch, object_info->orientation.roll,
 object_info->rotation.yaw_rate, object_info->rotation.pitch_rate,
 object_info->rotation.roll_rate);

fclose (ptr_to_display_file);
}
else
printf ("CANNOT OPEN DISPLAY FILE IN SEND_FUPDATE\n");
}



/***************************************************************************/
/* DATE: 09/04/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: sensor_check                                                       */
/* MODULE_NUMBER: 2.18                                                       */
/* DESCRIPTION: This function is used to check whether a sensor contact      */
/*              is made prior to the next scheduled turnpoint.               */
/* ALGORITHM:   see discussion in Thesis                                     */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/* PASSED VARIABLES: struct argument_type argument                           */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: extern struct driver* driver                     */
/*                          extern struct linked_list* master_ob_list        */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN:                                                            */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: almost all the events call this function                 */
/* ORDER OF: O(1)                                                            */
/* AUTHOR: Rob Rizza                                                         */
```

```
/* HISTORY: none                                                              */
/****************************************************************************/
void sensor_check (event_argument)
struct event_args *event_argument;
{
int num_objs = 0, num_vehicles = 0, i = 0, j = 0, sensor_contact_found = 0,
    valid_contact1 = 0, valid_contact2 = 0, contact1 = 0, contact2 = 0, sensing_range;

double curr_time, curr_x_coord_other_object, curr_y_coord_other_object,
       term_under_radical1, term_under_radical2, a, b, c1, c2, event_time,
       diff_in_curr_x_coords, diff_in_curr_y_coords, diff_in_curr_x_vels,
       diff_in_curr_y_vels, time_at_next_routept1, time_at_next_routept2,
       sensor_contact_time1, sensor_contact_time2, range1, range2, time_to_event;

double *time_ptr = NULL;

struct location_type *next_routept = NULL;
struct object_attributes *other_object = NULL;
struct object_attributes *object1 = NULL;
struct object_attributes *object2 = NULL;

extern struct linked_list *master_obj_list;
extern struct driver *simulation_driver;

num_vehicles = ll_length (master_obj_list);

if (event_argument->object1->velocity.x_velocity != 0.0 ||
    event_argument->object1->velocity.y_velocity != 0.0 ||
    event_argument->object1->velocity.z_velocity != 0.0)
{
time_at_next_routept1 = calc_time_at_next_routept (event_argument->object1);
event_time = time_at_next_routept1;

for (i = 1; i <= num_vehicles; i++)
{
other_object = (struct object_attributes*)ll_pop(master_obj_list);

if (event_argument->object1->object_id == other_object->object_id)
ll_insert (master_obj_list, other_object);

else
{
curr_time = event_argument->object1->current_time;

curr_x_coord_other_object = other_object->location.x_coord +
 ((curr_time - other_object->current_time)
     * other_object->velocity.x_velocity);
curr_y_coord_other_object = other_object->location.y_coord +
 ((curr_time - other_object->current_time)
     * other_object->velocity.y_velocity);

diff_in_curr_x_coords = event_argument->object1->location.x_coord -
curr_x_coord_other_object;
diff_in_curr_y_coords = event_argument->object1->location.y_coord -
curr_y_coord_other_object;

diff_in_curr_x_vels = event_argument->object1->velocity.x_velocity -
      other_object->velocity.x_velocity;
diff_in_curr_y_vels = event_argument->object1->velocity.y_velocity -
      other_object->velocity.y_velocity;


       /********** QUADRATIC EQUATION IS (t): t = (-b +- sqrt (bb - 4ac))/2a **********/

range1 = (double)get_sensor_range (event_argument->object1);
```

```
range2 = (double)get_sensor_range (other_object);

a = (diff_in_curr_x_vels * diff_in_curr_x_vels) +
    (diff_in_curr_y_vels * diff_in_curr_y_vels);
b = (2.0 * diff_in_curr_x_coords * diff_in_curr_x_vels) +
    (2.0 * diff_in_curr_y_coords * diff_in_curr_y_vels);
c1 = (diff_in_curr_x_coords * diff_in_curr_x_coords) +
     (diff_in_curr_y_coords * diff_in_curr_y_coords) -
     (range1 * range1);

c2 = (diff_in_curr_x_coords * diff_in_curr_x_coords) +
     (diff_in_curr_y_coords * diff_in_curr_y_coords) -
     (range2 * range2);

term_under_radical1 = (b * b) - (4.0 * a* c1);

if (fabs (term_under_radical1) < 0.0001)
term_under_radical1 = 0.0;

term_under_radical2 = (b * b) - (4.0 * a* c2);

if (fabs (term_under_radical2) < 0.0001)
term_under_radical2 = 0.0;

if (term_under_radic ]) >= 0.0 || term_under_radical2 >= 0.0)
{
if ((time_at_next_routept2 = calc_time_at_next_routept
 (other_object)) <= curr_time)
      time_at_next_routept2 = calc_time_at_nextnext_routept
 (other_object);
if (time_at_next_routept2 <= curr_time)
time_at_next_routept2 = time_at_next     opt1;

if (term_under_radical1 >= 0.0)
{
sensor_contact_time1 = (-1 * b -      (term_under_radical1)) /
   (2.0 * a) + curr_time;

if (sensor_contact_time1 < event_time
   && sensor_contact_time1 > (curr_time + 0.0000001)
   && sensor_contact_time1 <= time_at_next_routept1
   && line_of_sight (event_argument) == TRUE)
valid_contact1 = TRUE;
}

if (term_under_radical2 >= 0.0)
{
     /*  if (time_at_next_routept2 == other_object->current_time)
time_at_next_routept2 = time_at_next_routept1;  */

sensor_contact_time2 = (-1 * b - sqrt (term_under_radical2)) /
   (2.0 * a) + curr_time;

if (sensor_contact_time2 < event_time
   && sensor_contact_time2 > (curr_time + 0.0000001)
   && sensor_contact_time2 <= time_at_next_routept2
   && sensor_contact_time2 <= (time_at_next_routept1 + 0.0000001)
   && line_of_sight (event_argument) == TRUE)
valid_contact2 = TRUE;
}

if (valid_contact1 == TRUE && valid_cont     TRUE)
{
object1 = event_argument->object1;
object2 = other_object;
```

```
if (sensor_contact_time1 == sensor_contact_time2)
{
contact1 = TRUE;
contact2 = TRUE;

if (range1 == 0.0 && range2 == 0.0)
sensing_range = 0;
else
sensing_range = 1;

event_time = sensor_contact_time1;
}

if (sensor_contact_time1 > sensor_contact_time2)
{
contact1 = FALSE;
contact2 = TRUE;

event_time = sensor_contact_time2;
}

if (sensor_contact_time1 < sensor_contact_time2)
{
contact1 = TRUE;
contact2 = FALSE;

event_time = sensor_contact_time1;
}
}
else if (valid_contact1 == TRUE && valid_contact2 == FALSE)
{
contact1 = TRUE;
contact2 = FALSE;

event_time = sensor_contact_time1;
object1 = event_argument->object1;
object2 = other_object;
}
else if (valid_contact1 == FALSE && valid_contact2 == TRUE)
{
contact1 = FALSE;
contact2 = TRUE;

event_time = sensor_contact_time2;
object1 = event_argument->object1;
object2 = other_object;
}
}
ll_insert (master_obj_list, other_object);
valid_contact1 = FALSE;
valid_contact2 = FALSE;
}
}

if (contact1 == TRUE && contact2 == TRUE && sensing_range == 0 &&
    difference_in_altitude (event_argument) == 0.0)
{
event_argument->event_time = event_time;
event_argument->object1 = object1;
event_argument->object2 = object2;

add_event_coords_to_route (event_argument);

if ((time_ptr = (double*)malloc(sizeof(curr_time))) == NULL)
```

```
printf ("CANNOT MALLOC TIME_PTR IN SENSOR_CHECK\n");

*time_ptr = event_argument->event_time;

schedule_event (simulation_driver, time_ptr, collision_distance_reached, event_argument);
}

else if (contact1 == TRUE || contact2 == TRUE)
{
if (contact1 == TRUE && contact2 == TRUE)
{
time_to_event = event_argument->object1->current_time - event_time;
add_new_routepoint (event_argument->object1, time_to_event);
add_new_routepoint (event_argument->object1, time_to_event);
}
else
{
time_to_event = event_time - event_argument->object1->current_time;
add_new_routepoint (event_argument->object1, time_to_event);
}

event_argument->event_time = event_time;
event_argument->object1 = object1;
event_argument->object2 = object2;

if ((time_ptr = (double*)malloc(sizeof(curr_time))) == NULL)
printf ("CANNOT MALLOC TIME_PTR IN SENSOR_CHECK\n");

*time_ptr = event_argument->event_time;

if (contact2 == TRUE)
schedule_event (simulation_driver, time_ptr, entered_sensor_range,
 event_argument);
if (contact1 == TRUE && event_argument->object1->object_type != MISSLE)
schedule_event (simulation_driver, time_ptr, made_sensor_contact,
 event_argument);
else if (contact1 == TRUE && event_argument->object1->object_type == MISSLE)
schedule_event (simulation_driver, time_ptr, ordnance_reached_target,
 event_argument);
}

else
if (ll_isempty (event_argument->object1->route_data) != TRUE)
{
next_routept = (struct location_type*)ll_pop
 (event_argument->object1->route_data);

ll_insert (event_argument->object1->route_data, next_routept);

if ((time_ptr = (double*)malloc(sizeof(curr_time))) == NULL)
printf ("CANNOT MALLOC IN CASE 0x01 OF SENSOR_CHECK\n");

*time_ptr = time_at_next_routept1;

event_argument->object2 = NULL;
event_argument->event_time = event_time;

schedule_event (simulation_driver, time_ptr, reached_turnpoint,
 event_argument);
}
}
}
```

```
/*****************************************************************************/
/* DATE: 09/11/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: terminate_vehicle                                                  */
/* MODULE_NUMBER: 2.19                                                       */
/* DESCRIPTION: This function is used to terminate an object and any         */
/*              associated events                                            */
/* ALGORITHM: - send the terminator identifier, the vehicle_id and time to   */
/*              terminate to the display driver                              */
/*            - delete scheduled events that are associated with the         */
/*              identified vehicle_id                                        */
/* PASSED VARIABLES: struct object_attributes* object_info                   */
/* RETURNS: struct linked_list* deleted_events                               */
/* GLOBAL VARIABLES PASSED: simulation_driver, master_obj_list               */
/* GLOBAL VARIABLES CHANGED: master_obj_list                                 */
/* FILES READ: none                                                          */
/* FILES WRITTEN: display.c                                                  */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none*                                                    */
/* MODULES CALLED: delete_event                                              */
/* CALLING MODULES: main                                                     */
/* ORDER OF: This function is of order O(n) where n is the number of objects*/
/*           in the master_obj_list                                          */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
int delete_object (); /* struct object_attributes* object_info, int* vehicle_id */

struct linked_list *terminate_objects (event_argument)
struct event_args *event_argument;
{
extern struct linked_list *master_obj_list;
extern struct driver *simulation_driver;
struct linked_list *deleted_events = NULL;
struct driver_data *event_data;
struct event_args *deleted_event_argument;
struct location_type *bogus_routept;

FILE *ptr_to_display_file;

if ((ptr_to_display_file = fopen ("display.c", "a")) != NULL)
{
fprintf (ptr_to_display_file, "33 %d %lf\n", event_argument->object1->object_id,
event_argument->object1->current_time + 0.1);
fclose (ptr_to_display_file);
}

deleted_events = delete_event (simulation_driver, event_argument->object1->object_id);
ll_delete (master_obj_list, delete_object, &(event_argument->object1->object_id));
while (ll_isempty (deleted_events) != TRUE)
{
event_data = ll_pop (deleted_events);
deleted_event_argument = event_data->func_arguments;

if (deleted_event_argument->object2 != NULL)
if (deleted_event_argument->object2->object_id ==
 event_argument->object1->object_id)
{
bogus_routept = ll_pop (deleted_event_argument->object1->route_data);
free (bogus_routept);
sensor_check (deleted_event_argument);
}
}
free (event_argument);
```

```
return deleted_events;

}

/*****************************************************************************/
/* DATE: 09/12/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: delete_vehicle                                                     */
/* MODULE_NUMBER: 2.19a                                                      */
/* DESCRIPTION: This function is used by ll_delete to delete a pointer to    */
/*              a object from the master_obj_list                            */
/* ALGORITHM: For as many items that there are in the list                   */
/*              - if the vehicle_id from the list matches the referenced id  */
/*                  then delete it from the list                             */
/* PASSED VARIABLES: struct object_attributes* object_info, int object_id    */
/* RETURNS: result                                                           */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES: ll_delete                                                */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
int delete_object (object_info, obj_id)
struct object_attributes *object_info;
int *obj_id;
{
int result;

if (object_info->object_id == *obj_id)
result = LL_DEL_YES | LL_STOP;
else
result =  LL_DEL_NO;

return result;
}



/*****************************************************************************/
/* DATE: 09/30/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: update_object_current_time                                         */
/* MODULE_NUMBER: 2.20                                                       */
/* DESCRIPTION: Simply updates the current time of the object                */
/* ALGORITHM: update the current object time to the current event time       */
/* PASSED VARIABLES: event_args *event_argument                              */
/* RETURNS:  none                                                            */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: none                                                      */
/* CALLING MODULES:                                                          */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/
```

```
void update_object_current_time (event_argument)
struct event_args *event_argument;
{
if (event_argument->object1 != NULL)
event_argument->object1->current_time = event_argument->event_time;
}



/*****************************************************************************/
/* DATE: 08/02/90                                                            */
/* VERSION: 0.0                                                              */
/* TITLE: update_position                                                    */
/* MODULE_NUMBER: 2.21                                                       */
/* DESCRIPTION: This function is used to extract the next route point from   */
/*              the route data linked list                                   */
/* ALGORITHM: pop the next route data point from the linked list             */
/*            extract and read the required data into the current location   */
/*            and orientation attributes of the vehicle                      */
/* PASSED VARIABLES: object_info                                             */
/* RETURNS: none                                                             */
/* GLOBAL VARIABLES PASSED: none                                             */
/* GLOBAL VARIABLES CHANGED: none                                            */
/* FILES READ: none                                                          */
/* FILES WRITTEN: none                                                       */
/* HARDWARE INPUT: none                                                      */
/* HARDWARE OUTPUT: none                                                     */
/* MODULES CALLED: calc_curr_orientation                                     */
/* CALLING MODULES: reach_turnpoint, reached_target, reached_destination,    */
/*                  reached_sensor_range                                     */
/* ORDER OF: This function is of order O(1)                                  */
/* AUTHOR: Rob Rizza                                                         */
/* HISTORY: none                                                             */
/*****************************************************************************/

void update_position (event_argument)
struct event_args *event_argument;
{
struct location_type *new_position_info = NULL;

if (ll_isempty (event_argument->object1->route_data) != TRUE)
{
new_position_info = (struct location_type*)ll_pop
(event_argument->object1->route_data);
event_argument->object1->location.x_coord = new_position_info->x_coord;
event_argument->object1->location.y_coord = new_position_info->y_coord;
event_argument->object1->location.z_coord = new_position_info->z_coord;

free(new_position_info);
calc_curr_orientation (event_argument->object1);
calc_curr_velocities (event_argument->object1);
update_object_current_time (event_argument);
send_fupdate (event_argument->object1);
}
if (ll_isempty (event_argument->object1->route_data) &&
(event_argument->object1->object_type == MISSLE))
terminate_objects (event_argument);
}
```

Appendix B. TESTING STRATEGIES, RESULTS and CODE

B.1 Testing Strategies

A bottom-up strategy was used to test the simulation software written for this thesis. Since the design was completed prior to the beginning of any coding, it was reasonably simple to identify those functions that did not require any outside input other than those variables passed as arguments. Thus, since these functions made no function calls from within their code, they made up the lowest level of functions which were tested. Figure B.1 illustrates the order in which testing took place. The lowest step functions must be developed and tested prior to moving to the next higher step.

Each function tested was tested using two strategies. First, all known boundary values were tested, as well as any value which was determined to be a possible problem (zero is a good example of a problem value). Second, to the extent to which it could be determined, all branches were exercised within each function.

As integration took place, the same two strategies were used as in each function test. However, it should be noted that although I used, for the most part, the same boudary or problem values previously found, these may or may not have been boundary or problem values once integration took place. Also, as integration took place, determining all possible paths quickly became a real problem. Thus, it is likely that only a sampling of paths were tested during the integration phase.

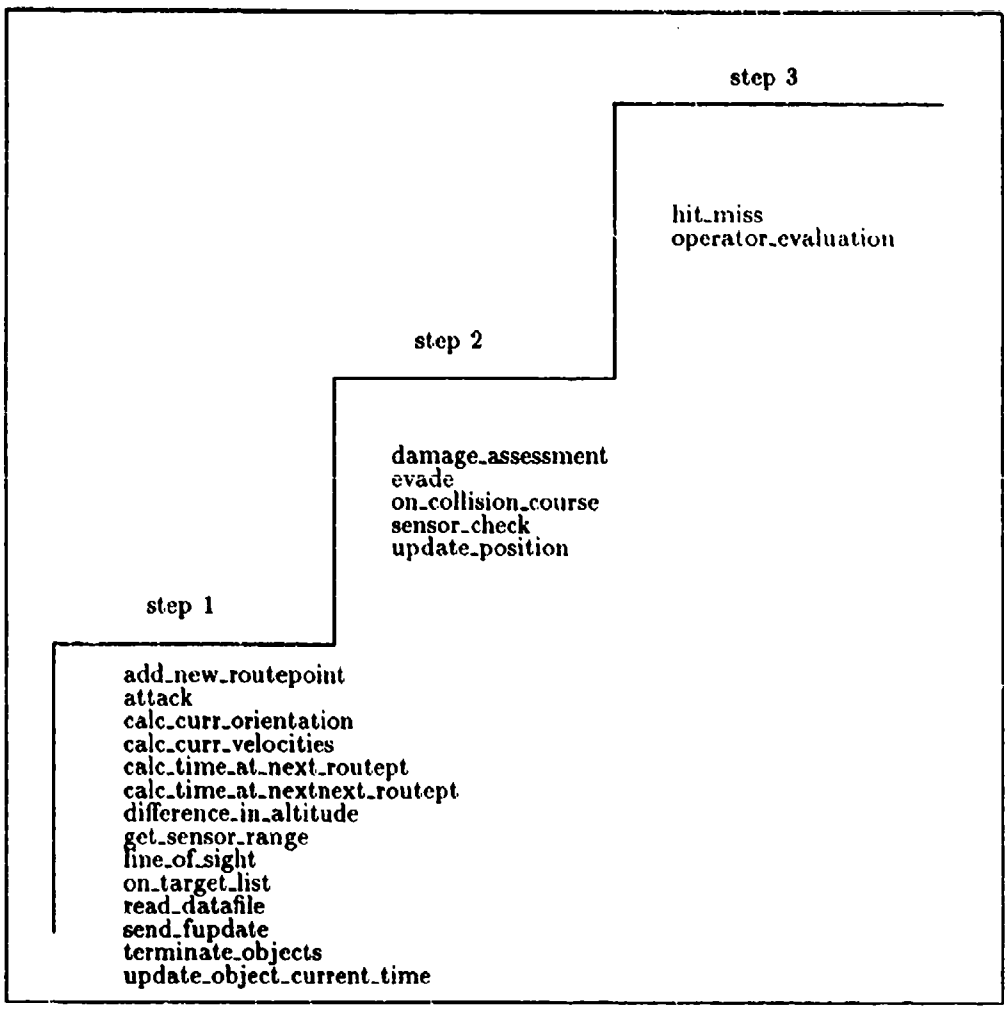


Figure B.1. Function Development and Testing Staircase

B.2 Test Results and Code

The objective of any software testing is to find errors (15:191). True to the stated objective many errors were found and corrected. However, the test results given here are not from the tests where errors were found. The results given are those found after needed corrections were made. The reason for inclusion of this information here is to first, show what values were used in the testing process, and second, to provide repeatable data which can be used in future testing or validation.

Please note, the following test code was written for the most part in ANSI C to run using Microsoft's Quick C; thus if used on another system some slight modifications may be needed.

B.2.1 Test 1, add_new_routepoint The following is a copy of the code used to test the add new routepoint function. There were two concerns when testing this function. First, were the calculation being done correctly and, second were the routepoints being properly placed into the object's route data. Initial coordinates, velocity vectors, or the "in x seconds" quantities were varied during each test run. No problems were encountered.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include <stdio.h>

void *add_new_routepoint (struct object_attributes *object_info, double in_x_seconds);

void main()
{
struct object_attributes F15;
struct location_type *new_point;
FILE *ptr_to_test_file;

F15.location.x_coord = 0;
F15.location.y_coord = 0;
F15.location.z_coord = 0;

F15.velocity.x_velocity = 150;
F15.velocity.y_velocity = 150;
F15.velocity.z_velocity = 0;

F15.route_data = ll_make (LIFO);
```

```
add_new_routepoint (&F15, 10);

new_point = ll_pop (F15.route_data);

ptr_to_test_file = fopen("test1.res","a");

fprintf (ptr_to_test_file, "x = %lf\t y = %lf\t z = %lf\n", new_point->x_coord, new_point->y_coord,
new_point->z_coord);

fclose (ptr_to_test_file);
}

void *add_new_routepoint (object_info, in_x_seconds)
struct object_attributes *object_info;
double in_x_seconds;
{
struct location_type *new_next_pt = NULL;

if ((new_next_pt = (struct location_type*)malloc(sizeof(struct location_type)))==NULL)
return NULL;

new_next_pt->x_coord = (object_info->location.x_coord + (in_x_seconds * object_info->velocity.x_velocity));
new_next_pt->y_coord = (object_info->location.y_coord + (in_x_seconds * object_info->velocity.y_velocity));
new_next_pt->z_coord = (object_info->location.z_coord + (in_x_seconds * object_info->velocity.z_velocity));

ll_insert (object_info->route_data, new_next_pt);
}

/************************** sample output ************************************/

x = 1500.000000  y = 1500.000000  z = 0.000000
```

B.2.2 Test 2, calc_curr_orientation The following is a copy of the code used to test the calculate current orientation function. Keeping the initial point coordinates at 0, 0, 0, the next point was varied such that orientations returned were from all quadrants including angles of 0, 90, 180, and 270. (see the sample output following the test code.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include<math.h>
#include <stdio.h>

void calc_curr_orientation (struct object_attributes *object_info);

void main()
{
struct object_attributes F15;
struct location_type point1;
FILE *ptr_to_test_file;
```

```
F15.location.x_coord = 0;
F15.location.y_coord = 0;
F15.location.z_coord = 0;

F15.velocity.x_velocity = 0;
F15.velocity.y_velocity = 0;
F15.velocity.z_velocity = 0;

F15.route_data = ll_make (LIFO);

point1.x_coord = 0;
point1.y_coord = 100;
point1.z_coord = 100;

ll_insert (F15.route_data, &point1);

calc_curr_orientation (&F15);

ptr_to_test_file = fopen("test2.res", "a");

fprintf (ptr_to_test_file, "yaw = %lf\t pitch = %lf\t roll = %lf\n", F15.orientation.yaw, F15.orientation.pitch,
 F15.orientation.roll);

fclose(ptr_to_test_file);
}

void calc_curr_orientation (object_info)
struct object_attributes *object_info;
{
double delta_x, delta_y, delta_z, distance, angle, pitch;
struct location_type *next_route_point = NULL;

if (ll_isempty (object_info->route_data) != 1)
{
next_route_point = (struct location_type*)ll_pop (object_info->route_data);
delta_x = next_route_point->x_coord - object_info->location.x_coord;
delta_y = next_route_point->y_coord - object_info->location.y_coord;
delta_z = next_route_point->z_coord - object_info->location.z_coord;

angle = atan2 (delta_y, delta_x) * 360 / (2 * 3.14159);

if (angle < 0.0)
angle = 360 + angle;

if (angle >= 0.0 && angle <= 90.0)
object_info->orientation.yaw = 90.0 - angle;
else if (angle > 90.0 && angle <= 180.0)
object_info->orientation.yaw = 360.0 - (angle - 90.0);
else if (angle > 180.0 && angle <= 270.0)
object_info->orientation.yaw = 270.0 - (angle - 180.0);
else
object_info->orientation.yaw = 180 - (angle - 270.0);

pitch = atan2 (delta_z, (distance = sqrt ((delta_x*delta_x)+(delta_y*delta_y)))) * 360 / (2 * 3.14159);

object_info->orientation.pitch = pitch;
object_info->orientation.roll = object_info->orientation.roll;

ll_insert (object_info->route_data, next_route_point);
}
else
{
object_info->orientation.pitch = 0.0;
object_info->orientation.roll = 0.0;
}
```

```
}

/************************************ sample output ***************************************/

yaw = 44.999962     pitch = 35.264419      roll = 0.000000
yaw = 90.000000     pitch = 45.000038      roll = 0.000000
yaw = 135.000038    pitch = 35.264419      roll = 0.000000
yaw = 180.000076    pitch = 45.000038      roll = 0.000000
yaw = 225.000114    pitch = 35.264419      roll = 0.000000
yaw = 269.999848    pitch = 45.000038      roll = 0.000000
yaw = 314.999886    pitch = 35.264419      roll = 0.000000
yaw = 359.999924    pitch = 45.000038      roll = 0.000000
```

B.2.3 Test 3, calc_curr_velocities The following code was used to test the calculate current velocities function. Like calculate current orientation, routepoints were varied such that all quadrants were represented. See the sample output.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include <math.h>
#include <stdio.h>

void calc_curr_velocities (struct object_attributes *object_info);

void main()
{
struct object_attributes F15;
struct location_type point1;
FILE *ptr_to_test_file;

F15.location.x_coord = 0;
F15.location.y_coord = 0;
F15.location.z_coord = 0;

F15.velocity.x_velocity = 150;
F15.velocity.y_velocity = 150;
F15.velocity.z_velocity = 0;

F15.route_data = ll_make (LIFO);

point1.x_coord = 0;
point1.y_coord = 100;
point1 z_coord = 10;

ll_insert (F15.route_data, &point1);

calc_curr_velocities (&F15);

ptr_to_test_file = fopen("test3.res", "a");

fprintf (ptr_to_test_file, "x_vel = %lf\t y_vel = %lf\t z_vel = %lf\n", F15.velocity.x_velocity,
F15.velocity.y_velocity, F15.velocity.z_velocity);

fclose(ptr_to_test_file);
}
```

```
void calc_curr_velocities (object_info)
struct object_attributes *object_info;
{
struct location_type *next_route_point = NULL;
double delta_x, delta_y, delta_z, slope_angle, horizontal_vel_vector,
       time_to_next_route_point, distance_to_next_route_point;

if (ll_isempty (object_info->route_data) != 1)
{
next_route_point = (struct location_type*)ll_pop (object_info->route_data);
horizontal_vel_vector = sqrt ((object_info->velocity.x_velocity * object_info->velocity.x_velocity) +
 (object_info->velocity.y_velocity * object_info->velocity.y_velocity));

delta_x = next_route_point->x_coord - object_info->location.x_coord;
delta_y = next_route_point->y_coord - object_info->location.y_coord;
delta_z = next_route_point->z_coord - object_info->location.z_coord;

slope_angle = atan2 (delta_y, delta_x);
distance_to_next_route_point = sqrt ((delta_x * delta_x) + (delta_y * delta_y));
time_to_next_route_point = distance_to_next_route_point / horizontal_vel_vector;

object_info->velocity.x_velocity = horizontal_vel_vector * cos(slope_angle);
object_info->velocity.y_velocity = horizontal_vel_vector * sin(slope_angle);
object_info->velocity.z_velocity = delta_z / time_to_next_route_point;

ll_insert (object_info->route_data, next_route_point);
}
else
{
object_info->velocity.x_velocity = 0.0;
object_info->velocity.y_velocity = 0.0;
object_info->velocity.z_velocity = 0.0;
}
}

/************************************ sample output ***************************************/

x_vel = 150.000000       y_vel = 150.000000      z_vel = 15.000000
x_vel = 212.132034  y_vel = 0.000000  z_vel = 21.213203
x_vel = 150.000000  y_vel = -150.000000  z_vel = 15.000000
x_vel = 0.000000  y_vel = -212.132034  z_vel = 21.213203
x_vel = -150.000000  y_vel = -150.000000  z_vel = 15.000000
x_vel = -212.132034  y_vel = 0.000000  z_vel = 21.213203
x_vel = -150.000000  y_vel = 150.000000  z_vel = 15.000000
x_vel = 0.000000  y_vel = 212.132034  z_vel = 21.213203
x_vel = 0.000000  y_vel = 212.132034  z_vel = 21.213203
```

B.2.4 Test 4, calc_time_at_next_routept The following code was used to test the calculate time at next routepoint function. Values for the next routepoint locations varied but included values in all quadrants as well as the next routepoint being the same as the current routepoint location.

```
#include "sim_func.h"
#include "sim_stru.h"
```

```
#include "ll.h"
#include <stdio.h>
#include <math.h>

double calc_time_at_next_routept (struct object_attributes *object_info);

void main ()
{
double time_at_nextpt;
struct object_attributes F15, M29;
struct location_type point_1, point_2, point_3;
FILE *ptr_to_test_file;

F15.location.x_coord = 0;
F15.location.y_coord = 0;
F15.location.z_coord = 0;

F15.velocity.x_velocity = 150;
F15.velocity.y_velocity = 150;
F15.velocity.z_velocity = 0;

F15.current_time = 0;

point_1.x_coord = -300;
point_1.y_coord = 300;
point_1.z_coord = 30;

point_2.x_coord = 0;
point_2.y_coord = -5;
point_2.z_coord = 0;


F15.route_data = ll_make (LIFO);

ll_insert (F15.route_data, &point_2);
ll_insert (F15.route_data, &point_1);

time_at_nextpt = calc_time_at_next_routept(&F15);

ptr_to_test_file = fopen("test4.res","a");

fprintf (ptr_to_test_file, "time at next routepoint = %lf\n", time_at_nextpt);

fclose(ptr_to_test_file);
}

double calc_time_at_next_routept (object_info)
struct object_attributes *object_info;
{
double delta_x, delta_y, delta_z, distance_traveled, time_at_next_routept, total_vel_vector;
struct location_type *next_routept = NULL;
int event;

time_at_next_routept = object_info->current_time;
if (ll_isempty (object_info->route_data) != 1)
{
next_routept = (struct location_type*)ll_pop (object_info->route_data);
delta_x = object_info->location.x_coord - next_routept->x_coord;
delta_y = object_info->location.y_coord - next_routept->y_coord;
delta_z = object_info->location.z_coord - next_routept->z_coord;

ll_insert (object_info->route_data, next_routept);

distance_traveled = sqrt ((delta_x*delta_x) + (delta_y*delta_y) + (delta_z*delta_z));
```

```
total_vel_vector = sqrt ((object_info->velocity.x_velocity * object_info->velocity.x_velocity) +
      (object_info->velocity.y_velocity * object_info->velocity.y_velocity) +
      (object_info->velocity.z_velocity * object_info->velocity.z_velocity));

if (total_vel_vector != 0.0)
time_at_next_routept = object_info->current_time + distance_traveled / total_vel_vector;
else
time_at_next_routept = object_info->current_time;

}
return time_at_next_routept;
}

/************************************************ sample output ***********************************/

time at next routepoint = 0.000000
time at next routepoint = 2.004994
time at next routepoint = 2.004994
time at next routepoint = 2.004994
time at next routepoint = 2.004994
```

B.2.5 Test 5, calc_time_at_nextnext_routept The following code was used to test the calculate time at nextnext routepoint function. The same strategy used in testing calculate time at next routepoint was used to test this function.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include <stdio.h>
#include <math.h>

double calc_time_at_nextnext_routept (struct object_attributes *object_info);

void* main ()
{
double time_at_nextpt;
struct object_attributes F15, M29;
struct location_type point_1, point_2, point_3;
FILE *ptr_to_test_file;

F15.location.x_coord = 0;
F15.location.y_coord = 0;
F15.location.z_coord = 0;

F15.velocity.x_velocity = 150;
F15.velocity.y_velocity = 150;
F15.velocity.z_velocity = 0;

F15.current_time = 0;

point_1.x_coord = 300;
point_1.y_coord = 300;
point_1.z_coord = 30;

point_2.x_coord = 300;
point_2.y_coord = 300;
```

```
point_2.z_coord = 30;


F15.route_data = ll_make (LIFO);

ll_insert (F15.route_data, &point_2);
ll_insert (F15.route_data, &point_1);

time_at_nextpt = calc_time_at_nextnext_routept(&F15);

ptr_to_test_file = fopen("test5.res","a");

fprintf (ptr_to_test_file, "time at next routepoint = %lf\n", time_at_nextpt);

fclose(ptr_to_test_file);
}

double calc_time_at_nextnext_routept (object_info)
struct object_attributes *object_info;
{

double delta_x, delta_y, delta_z, distance_traveled, time_at_next_routept, total_vel_vector;
struct location_type *next_routept = NULL, *nextnext_routept = NULL;
int event;

time_at_next_routept = object_info->current_time;

if (ll_isempty (object_info->route_data) != 1)
{
next_routept = (struct location_type*)ll_pop (object_info->route_data);
delta_x = object_info->location.x_coord - next_routept->x_coord;
delta_y = object_info->location.y_coord - next_routept->y_coord;
delta_z = object_info->location.z_coord - next_routept->z_coord;

distance_traveled = sqrt ((delta_x*delta_x) + (delta_y*delta_y) + (delta_z*delta_z));

total_vel_vector = sqrt ((object_info->velocity.x_velocity * object_info->velocity.x_velocity) +
      (object_info->velocity.y_velocity * object_info->velocity.y_velocity) +
      (object_info->velocity.z_velocity * object_info->velocity.z_velocity));

if (total_vel_vector != 0.0)
time_at_next_routept = object_info->current_time + distance_traveled / total_vel_vector;
else
time_at_next_routept = object_info->current_time;

}
if (ll_isempty (object_info->route_data) != 1)
{
nextnext_routept = (struct location_type*)ll_pop (object_info->route_data);
delta_x = next_routept->x_coord - nextnext_routept->x_coord;
delta_y = next_routept->y_coord - nextnext_routept->y_coord;
delta_z = next_routept->z_coord - nextnext_routept->z_coord;

distance_traveled = sqrt ((delta_x*delta_x) + (delta_y*delta_y) + (delta_z*delta_z));

total_vel_vector = sqrt ((object_info->velocity.x_velocity * object_info->velocity.x_velocity) +
      (object_info->velocity.y_velocity * object_info->velocity.y_velocity) +
      (object_info->velocity.z_velocity * object_info->velocity.z_velocity));

ll_insert (object_info->route_data, nextnext_routept);
ll_insert (object_info->route_data, next_routept);

if (total_vel_vector != 0.0)
time_at_next_routept = time_at_next_routept + distance_traveled / total_vel_vector;
else
```

```
time_at_next_routept = time_at_next_routept;

return time_at_next_routept;
}

if (next_routept != NULL && nextnext_routept == NULL)
ll_insert (object_info->route_data, next_routept);

return time_at_next_routept;
}

/********************************** sample output ********************************/


time at next routepoint = 0.000000
time at next routepoint = 4.009988
time at next routepoint = 4.009988
time at next routepoint = 4.009988
time at next routepoint = 4.009988
time at next routepoint = 2.004994
```

B.2.6 Test 6, attack The following code was used to test the attack function. The major concerns in test this function were,whether the missle object was getting created properly (i.e. a type 30 message was passed to the dislpay), and whether the missle's routepoints were getting properly calculated and placed into the missle's route data.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include <stdio.h>
#include <math.h>

void attack (struct event_args *event_argument);

void main ()
{
struct object_attributes F15;
struct location_type point_1, point_2, point_3;
struct event_args *event_argument;

F15.velocity.x_velocity = 10;
F15.velocity.y_velocity = 10;
F15.velocity.z_velocity = 0;

F15.object_type = 1;
F15.object_id = 123;
F15.current_time = 0;

F15.location.x_coord = 0;
F15.location.y_coord = 0;
F15.location.z_coord = 0;
```

```
point_1.x_coord = 500;
point_1.y_coord = 500;
point_1.z_coord = 100;

F15.route_data = ll_make (LIFO);
ll_insert (F15.route_data, &point_1);

event_argument->object2 = &F15;
event_argument->event_time = 10;

attack(event_argument);
}

void attack (event_argument)
struct event_args* event_argument;
{
      /*  extern struct linked_list *master_obj_list;
extern struct driver *simulation_driver;
extern int highest_obj_id;                */

struct location_type *temp1, *temp2, *missle_routept1, *missle_routept2, *missle_routept3;
struct object_attributes *missle;
struct event_args *new_event_argument;
struct location_type *missle_routept;

FILE *ptr_to_test_file;
double *time_ptr;

if ((missle = (struct object_attributes*)malloc(sizeof(struct object_attributes))) != NULL)
{
missle->object_type = 3;
      /*  missle->object_id = ++highest_obj_id;  */
missle->current_time = event_argument->event_time;
missle->location.x_coord = event_argument->object1->location.x_coord;
missle->location.y_coord = event_argument->object1->location.y_coord;
missle->location.z_coord = event_argument->object1->location.z_coord;
missle->velocity.x_velocity = 1000.0;
missle->velocity.y_velocity = 0.0;
missle->velocity.z_velocity = 0.0;
missle->orientation.yaw = 0.0;
missle->orientation.pitch = 0.0;
missle->orientation.roll = 0.0;
missle->rotation.yaw_rate = 0.0;
missle->rotation.pitch_rate = 0.0;
missle->rotation.roll_rate = 0.0;
missle->sensors = NULL;
missle->target_list = NULL;
missle->armaments = NULL;
missle->defensive_systems = NULL;

temp1 = ll_pop (event_argument->object2->route_data);

missle->route_data = ll_make (LIFO);

missle_routept3 = (struct location_type*)malloc(sizeof(struct location_type));
missle_routept3->x_coord = temp1->x_coord;
missle_routept3->y_coord = temp1->y_coord;
missle_routept3->z_coord = temp1->z_coord;

missle_routept2 = (struct location_type*)malloc(sizeof(struct location_type));
missle_routept2->x_coord = event_argument->object2->location.x_coord +
   ((event_argument->event_time - event_argument->object2->current_time) *
   event_argument->object2->velocity.x_velocity);
missle_routept2->y_coord = event_argument->object2->location.y_coord +
```

```
   ((event_argument->event_time - event_argument->object2->current_time) *
   event_argument->object2->velocity.y_velocity);
missle_routept2->z_coord = event_argument->object2->location.z_coord +
   ((event_argument->event_time - event_argument->object2->current_time) *
   event_argument->object2->velocity.z_velocity);

if (fabs (missle_routept2->x_coord - missle_routept3->x_coord) < 0.001 &&
    fabs (missle_routept2->y_coord - missle_routept3->y_coord) < 0.001 &&
    fabs (missle_routept2->z_coord - missle_routept3->z_coord) < 0.001)
{
temp2 = ll_pop (event_argument->object2->route_data);
missle_routept3->x_coord = temp2->x_coord;
missle_routept3->y_coord = temp2->y_coord;
missle_routept3->z_coord = temp2->z_coord;
ll_insert (missle->route_data, missle_routept3);
ll_insert (missle->route_data, missle_routept2);
ll_insert (event_argument->object2->route_data, temp2);
ll_insert (event_argument->object2->route_data, temp1);
}
else
{
ll_insert (event_argument->object2->route_data, temp1);
ll_insert (missle->route_data, missle_routept3);
ll_insert (missle->route_data, missle_routept2);
}
missle_routept1 = (struct location_type*)malloc(sizeof(struct location_type));
missle_routept1->x_coord = missle->location.x_coord;
missle_routept1->y_coord = missle->location.y_coord;
missle_routept1->z_coord = missle->location.z_coord;
ll_insert (missle->route_data, missle_routept1);

if ((new_event_argument = (struct event_args*)malloc(sizeof(struct event_args))) == NULL)
printf ("CANNOT MALLOC NEW_EVENT_ARGUMENT IN ATTACK");
new_event_argument->object1 = missle;
new_event_argument->object2 = event_argument->object2;
new_event_argument->event_time = event_argument->event_time;

      /*  ll_insert (master_obj_list, missle);   */

ptr_to_test_file = fopen ("test6.res", "a");
fprintf (ptr_to_test_file, "30 %d %d\n", missle->object_id, missle->object_type);
while (ll_isempty(new_event_argument->object1->route_data) != 1)
{
missle_routept = ll_pop(new_event_argument->object1->route_data);
fprintf (ptr_to_test_file, "x coord = %lf\t y coord = %lf\t z coord =
  %lf\n", missle_routept->x_coord,
missle_routept->y_coord, missle_routept->z_coord);
}
fclose (ptr_to_test_file);

      /*  if ((time_ptr = (double*)malloc(sizeof(double))) == NULL)
printf ("CANNOT MALLOC TIME_PTR IN ATTACK");
*time_ptr = event_argument->event_time;

schedule_event (simulation_driver, time_ptr, ordnance_released, new_event_argument); */
}
else
printf ("CANNOT MALLOC MISSLE IN ATTACK");
}

/************************************ sample output ************************************/

30 0 3
x coord = 0.000000  y coord = 0.000000  z coord = 0.000000
x coord = 100.000000  y coord = 100.000000  z coord = 0.000000
```

```
x coord = 500.000000  y coord = 500.000000  z coord = 100.000000
30 0 3
x coord = 0.000000  y coord = 0.000000  z coord = 0.000000
x coord = 100.000000  y coord = 100.000000  z coord = 0.000000
x coord = 500.000000  y coord = 500.000000  z coord = 100.000000
```

B.2.7 Test 7, difference_in_altitude The following code was used to test the difference in altitude function. Z coordinate values above, below and, equal to each other were tested.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include <stdio.h>
#include <math.h>

double difference_in_altitude (struct event_args *event_argument);

void main ()
{
struct object_attributes F15, M29;
struct event_args *event_argument;
double difference;
FILE *ptr_to_test_file;

F15.location.z_coord = 0;
F15.current_time = 10;

M29.location.z_coord = 5;
M29.current_time =0;
M29.velocity.z_velocity = 0;

event_argument->object1 = &F15;
event_argument->object2 = &M29;

difference = difference_in_altitude(event_argument);

ptr_to_test_file = fopen("test7.res", "a");

fprintf (ptr_to_test_file, "difference = %lf\n", difference);

fclose (ptr_to_test_file);
}

double difference_in_altitude (event_argument)
struct event_args *event_argument;
{

double difference, curr_time;

curr_time = event_argument->object1->current_time;

if ((difference = fabs (event_argument->object1->location.z_coord -
    (event_argument->object2->location.z_coord +
    ((curr_time - event_argument->object2->current_time) *
    event_argument->object2->velocity.z_velocity)))) <= 5.0)
```

```
return 0.0;
else
return difference;
}

/************************************ sample output ************************************/

difference = 0.000000
difference = 1000.000000
difference = 0.000000
```

B.2.8 Test 8, get_sensor_range The code used to test the get sensor range function follows. Sensor range values within the object's sensor list varied above and below the default value as well as equal to the default value.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include <stdio.h>
#include <math.h>

#define MISSLE 3

int get_sensor_range (struct object_attributes *object_info);

void* main ()
{
struct object_attributes F15;
struct sensors *sensor1, *sensor2, *sensor3;
FILE *ptr_to_test_file;
int range;

F15.sensors = ll_make (FIFO);

sensor1->range = 0;
sensor2->range = 0;
sensor3->range = 834;

ll_insert (F15.sensors, sensor1);
ll_insert (F15.sensors, sensor2);
ll_insert (F15.sensors, sensor3);

range = get_sensor_range (&F15);

ptr_to_test_file = fopen ("test8.res", "a");

fprintf (ptr_to_test_file, "range = %d\n", range);

fclose (ptr_to_test_file);
}


int get_sensor_range (object_info)
struct object_attributes *object_info;
{
int i, length, range = 833, temp_range = 0;   /* default sensor range, approx 1/2 mile */
```

```
struct sensors *sensor = NULL;

if (object_info->sensors != NULL)
{
length = ll_length (object_info->sensors);

for (i = 1; i <= length; i++)
{
sensor = (struct sensors*)ll_pop (object_info->sensors);
if (sensor->range > range)
range = sensor->range;
ll_insert (object_info->sensors, sensor);
}
}
else
if (object_info->object_type == MISSLE)
range = 0;

return range;
}

/********************************** sample output ***************************************/

range = 3000
range = 833
range = 834
```

B.2.9 Test 9, on_target_list The following code was used to test the on target list function. The M29 object type was varied, values used were 1, 2, 3, 4, and 5. See sample output for results.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include <stdio.h>
#include <math.h>

#define MISSLE 3

int on_target_list (struct object_attributes *object1, struct object_attributes *object2);

void* main ()
{
struct object_attributes F15, M29;
struct targets target1;
struct targets target2;
struct targets target3;
FILE *ptr_to_test_file;
int return_value;

M29.object_type = 5;
F15.target_list = ll_make (FIFO);

target1.target_type = 3;
target2.target_type = 2;
target3.target_type = 1;
```

```
ll_insert (F15.target_list, &target1);
ll_insert (F15.target_list, &target2);
ll_insert (F15.target_list, &target3);

return_value = on_target_list (&F15, &M29);

ptr_to_test_file = fopen ("test10.res", "a");

fprintf (ptr_to_test_file, "return value = %d\n", return_value);

fclose (ptr_to_test_file);
}


int on_target_list (object1, object2)
struct object_attributes *object1;
struct object_attributes *object2;
{
int num_targets, i, return_value = 0;
struct targets *target;

if (object1->target_list != NULL)
{
num_targets = ll_length (object1->target_list);

for (i = 1; i <= num_targets; i++)
{
target = ll_pop (object1->target_list);

if (target->target_type == object2->object_type)
return_value = 1;

ll_insert (object1->target_list, target);
}
}
return return_value;
}

/************************************ sample output ************************************/

return value = 1
return value = 1
return value = 1
return value = 0
return value = 0
```

B.2.10 Test 10, send_fupdate The following code was used to test the send file update function.

```
#include "sim_func.h"
#include "sim_stru.h"
#include "ll.h"
#include <stdio.h>

void send_fupdate (struct object_attributes *object_info);
```

```
void* main ()
{
struct object_attributes F15;

F15.object_id = 0;
F15.current_time = 2;
F15.location.x_coord = 5;
F15.location.y_coord = 5;
F15.location.z_coord = 5;
F15.velocity.x_velocity = 10;
F15.velocity.y_velocity = 10;
F15.velocity.z_velocity = 10;
F15.orientation.yaw = 6;
F15.orientation.pitch = 6;
F15.orientation.roll = 6;
F15.rotation.yaw_rate = 7;
F15.rotation.pitch_rate = 7;
F15.rotation.roll_rate = 7;

send_fupdate (&F15);
}

void send_fupdate (object_info)
struct object_attributes *object_info;
{
FILE *ptr_to_display_file;

if ((ptr_to_display_file = fopen("display.c", "a")) != NULL)
{
fprintf (ptr_to_display_file, "31 %d %lf %.2lf %.2lf %.2lf %lf %lf %lf %lf %lf %lf %lf %lf %lf\n",
object_info->object_id, object_info->current_time, object_info->location.x_coord, object_info->location.y_coord,
object_info->location.z_coord, object_info->velocity.x_velocity,
object_info->velocity.y_velocity, object_info->velocity.z_velocity,
object_info->orientation.yaw, object_info->orientation.pitch,
object_info->orientation.roll, object_info->rotation.yaw_rate, object_info->rotation.pitch_rate,
 object_info->rotation.roll_rate);

fclose (ptr_to_display_file);
}
else
printf ("CANNOT OPEN DISPLAY FILE IN SEND_FUPDATE\n");
}


/************************************ sample output ************************************/

31 0 2.000000 5.00 5.00 5.00 10.000000 10.000000 10.000000 6.000000 6.000000 6.000000 7.000000 7.000000 7.000000
```

B.2.11 Other Level One Functions The following level one function's test code was not included here either because none was created since the function was extremely simple (i.e. the current line of sight function), or because they were tested during integration.

- terminate_objects
- update_object_current_time

- line_of_sight
- read_datafile

The following second and third level functions were tested during integration.

- operator_evaluation
- damage_assessment
- evade
- on_colision_course
- sensor_check
- update_position
- hit_miss

The following scenario datafile and simulation output can be used as a benchmark for future runs and verification of upper level function operation. Figure B.2 depicts the two dimensional representation of the scenario. At time t there are eight objects in the simulation, a flight of three approaching from the southwest, a single ship approaching from the northwest on an intersecting path with the flight of three, three tanks moving in a northwestly direction, and one other single ship moving north northwest. At time Δt later two of the flight of three has been destroyed as well as the single ship attacker. Now only one of the flight of three remains as well as the three tanks and the other single ship. By some other Δt later, the remaining single ship has turned due north to evade the other aircraft as the other aircraft flew by. On the last leg of the single ship's journey the single ship destroys the three tanks.

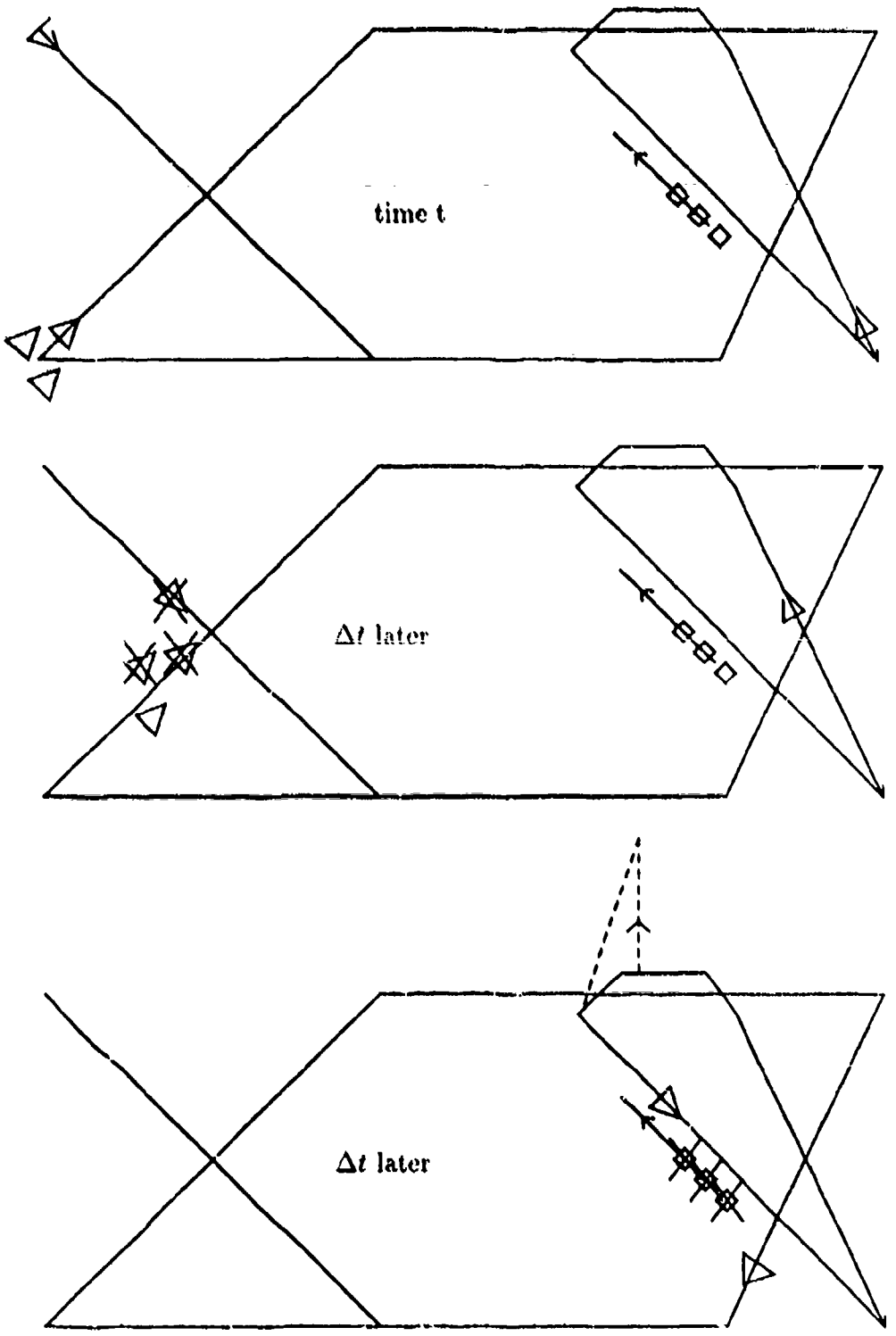


Figure B.2. Depiction of Benchmark Scenario

```
/*************************************** datafile ***************************************/

1 1 6 0 100 100 5 -20000 0 10 150 150 0 0 0 0 5 5 1 900 100 1000 5 0 0 10 38000 0 4000 48000 32000 4000
12000 32000 4000 -20000 0 10 1 1 2000 1 0 1 2 0 0 0 0
1 2 6 0 100 100 5 -20080 -80 10 150 150 0 0 0 0 5 5 1 900 100 1000 5 80 -80 10 38080 -80 4000 48080 32080
4000 11920 32080 4000 -20080 -80 10 1 1 2000 1 0 1 2 0 0 0 0
1 3 6 0 100 100 5 -19900 -300 10 150 150 0 0 0 0 5 5 1 900 100 1000 5 80 80 10 37920 80 4000 47920 31920
4000 12080 31920 4000 -19900 -300 10 1 1 2000 1 0 1 2 0 0 0 0
2 4 9 0 100 100 5 -20000 32000 10 150 -150 0 0 0 0 5 5 2 1000 100 1000 2 12000 0 3500 -20000 32000 8.5
 1 1 1300 1 0 0 0
2 5 9 0 100 100 5 48000 0 5.5 125 125 0 0 0 0 5 5 1 900 100 1000 6 48000 0 10 20000 30000 5000 18000
 34000 5000 31000 34000 5000 36000 30000 5000 48000 0 5.5 1 1 3000 1 0 1 4 0 0 0 0
4 6 6 0 100 100 5 32000 16000 2 -10 10 0 0 0 0 5 5 1 25 100 0 2 25000 24000 2 32000 16000 2 0 0 0 0
4 7 6 0 100 100 5 32070 15930 2 -10 10 0 0 0 0 5 5 1 25 100 0 2 25000 24000 2 32070 15930 2 0 0 0 0
4 8 6 0 100 100 5 32140 15860 2 -10 10 0 0 0 0 5 5 1 25 100 0 2 25000 24000 2 32140 15860 2 0 0 0 0


/*************************************** display file ***************************************/

32 1 f16
32 2 migl
32 3 missile
32 4 tank
32 5 truck
30 1 1
30 2 1
30 3 1
30 4 2
30 5 2
30 6 4
30 7 4
30 8 4
50
31 8 0.000000 32140.00 15860.00 2.00 -9.325600 10.631707 0.000000 318.744273 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
31 7 0.000000 32070.00 15930.00 2.00 -9.319192 10.637324 0.000000 318.778798 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
31 6 0.000000 32000.00 16000.00 2.00 -9.312661 10.643042 0.000000 318.813964 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
31 5 0.000000 48000.00 0.00 5.50 -65.653216 164.133041 27.325415 338.198486 8.787010
0.000000 0.000000 0.000000 0.000000 0.000000
31 4 0.000000 -20000.00 32000.00 8.50 150.000000 -150.000000 16.366406 135.000038 4.411747
 0.000000 0.000000 0.000000 0.000000 0.000000
31 3 0.000000 -19900.00 -300.00 10.00 149.438208 150.559696 18.644730 44.785773 5.022943
 0.000000 0.000000 0.000000 0.000000 0.000000
31 2 0.000000 -20080.00 -80.00 10.00 149.625470 150.373597 18.656426 44.857080 5.026078
 0.000000 0.000000 0.000000 0.000000 0.000000
31 1 0.000000 -20000.00 0.00 10.00 150.000000 150.000000 18.703125 44.999962 5.038594
  0.000000 0.000000 0.000000 0.000000 0.000000
31 1 100.000000 -5000.00 15000.00 1880.31 150.000000 150.000000 18.703125 44.999962
30 9 3
31 9 100.000000 -5000.00 15000.00 1880.31 0.000000 1000.000000 -117.585938 359.999924
-6.706388 0.000000 0.000000 0.000000 0.000000 0.000000
31 4 100.000000 -5000.00 17000.00 1645.14 150.000000 -150.000000 16.366406 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000 0.000000
31 4 100.153460 -4976.98 16976.98 1647.65 150.000000 -150.000000 16.366406 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000 0.000000
31 2 100.153460 -5094.49 14980.44 1878.51 149.625470 150.373597 18.656426 44.857080
 5.026078 0.000000 0.000000 0.000000 0.000000 0.000000
30 10 3
31 10 100.153460 -5094.49 14980.44 1878.51 58.755242 998.272418 -115.426690 3.368294
-6.584329 0.000000 0.000000 0.000000 0.000000 0.000000
31 4 100.611513 -4908.27 16908.27 1655.15 150.000000 -150.000000 16.366406 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000 0.000000
```

```
31 9 100.611513 -5000.00 15611.51 1808.41 0.000000 1000.000000 -117.585937 359.999924
-6.706388 0.000000 0.000000 0.000000 0.000000
31 4 100.770225 -4884.47 16884.47 1657.75 150.000000 -150.000000 16.366406 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000
31 3 100.813484 -4834.61 14878.45 1889.64 149.438208 150.559696 18.644730 44.785773
5.022943 0.000000 0.000000 0.000000 0.000000
30 11 3
31 11 100.813484 -4834.61 14878.45 1889.64 -21.681903 999.764920 -115.592854 358.757544
-6.593724 0.000000 0.000000 0.000000 0.000000
31 11 101.423153 -4847.83 15487.97 1819.17 -21.681903 999.764920 -115.592854 358.757544
-6.593724 0.000000 0.000000 0.000000 0.000000
31 9 102.000000 -5000.00 17000.00 1645.14 707.106781 -707.106781 77.151979 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000
31 10 102.153460 -4976.98 16976.98 1647.65 707.106781 -707.106781 77.151979 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000
31 10 102.250628 -4908.27 16908.27 1655.15 0.000000 0.000000 0.000000 135.000038 0.000000
0.000000 0.000000 0.000000 0.000000
33 10 102.350628
31 1 102.333333 -4650.00 15350.00 1923.95 150.000000 150.000000 18.703125 44.999962 5.038594
0.000000 0.000000 0.000000 0.000000
31 4 102.333333 -4650.00 16650.00 1683.33 150.000000 -150.000000 16.366406 135.000038 4.411747
0.000000 0.000000 0.000000 0.000000
30 12 3
31 12 102.333333 -4650.00 16650.00 1683.33 -0.000000 -1000.000000 185.095553 180.000076
10.486521 0.000000 0.000000 0.000000 0.000000
31 4 102.490372 -4626.44 16626.44 1685.90 150.000000 -150.000000 16.366406 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000
30 13 3
31 13 102.490372 -4626.44 16626.44 1685.90 -91.065945 -995.844864 181.696118 185.225013
10.298085 0.000000 0.000000 0.000000 0.000000
31 4 102.538496 -4619.23 16619.23 1686.69 150.000000 -150.000000 16.366406 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000
31 9 102.538496 -4619.23 16619.23 1686.69 707.106781 -707.106781 77.151979 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000
31 4 102.538496 -4619.23 16619.23 1686.69 150.000000 -150.000000 16.366406 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000
33 9 102.638496
33 4 102.638496
31 11 102.813484 -4877.98 16877.98 1658.45 707.106781 -707.106781 77.151979 135.000038
4.411747 0.000000 0.000000 0.000000 0.000000
31 11 103.135893 -4650.00 16650.00 1683.33 0.000000 0.000000 0.000000 135.000038 0.000000
0.000000 0.000000 0.000000 0.000000
33 11 103.235893
31 12 103.633333 -4650.00 15350.00 1923.95 707.106781 707.106781 88.167377 44.999962 5.038594
0.000000 0.000000 0.000000 0.000000
31 13 103.790372 -4744.83 15331.85 1922.10 705.341229 708.867936 87.947235 44.857080 5.026078
0.000000 0.000000 0.000000 0.000000
31 12 103.983356 -4402.50 15597.50 1954.81 707.106781 707.106781 88.167377 44.999962 5.038594
0.000000 0.000000 0.000000 0.000000
31 1 103.983356 -4402.50 15597.50 1954.81 150.000000 150.000000 18.703125 44.999962 5.038594
0.000000 0.000000 0.000000 0.000000
33 12 104.083356
33 1 104.083356
31 13 104.140395 -4497.94 15579.97 1952.89 705.341229 708.867936 87.947235 44.857080 5.026078
0.000000 0.000000 0.000000 0.000000
31 2 104.140395 -4497.94 15579.97 1952.89 149.625470 150.373597 18.656426 44.857080 5.026078
0.000000 0.000000 0.000000 0.000000
33 13 104.240395
33 2 104.240395
31 5 182.778555 36000.00 30000.00 5000.00 -138.039408 110.431526 0.000000 308.659689
0.000000 0.000000 0.000000 0.000000 0.000000
31 3 214.001495 12080.00 31920.00 4000.00 212.132034 0.000000 0.000000 90.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
31 5 219.000095 31000.00 34000.00 5000.00 -176.776695 0.000000 0.000000 269.999848 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
```

```
31 5 259.363771 23864.64 34000.00 5000.00 -176.776695 0.000000 0.000000 269.999848 0.000000
 0.000000 0.000000 0.000000 0.000000
31 5 259.363771 23864.64 34000.00 5000.00 0.000000 176.776695 0.000000 0.000000 0.000000
 0.000000 0.000000 0.000000 0.000000
31 5 319.363771 23864.64 44606.60 5000.00 -85.539065 -154.703162 0.000000 208.939352
 0.000000 0.000000 0.000000 0.000000 0.000000
31 3 382.952876 47920.00 31920.00 4000.00 -63.563157 -202.385091 0.000000 197.436048
0.000000 0.000000 0.000000 0.000000 0.000000
31 5 387.924759 18000.00 34000.00 5000.00 79.056942 -158.113883 0.000000 153.435002
0.000000 0.000000 0.000000 0.000000 0.000000
31 5 413.222980 20000.00 30000.00 5000.00 120.617965 -129.233534 -21.495845 136.974974
-6.933070 0.000000 0.000000 0.000000 0.000000
31 5 463.698013 26088.20 23476.93 3915.00 120.617965 -129.233534 -21.495845 136.974974
 -6.933070 0.000000 0.000000 0.000000 0.000000
30 14 3
31 14 463.698013 26088.20 23476.93 3915.00 531.180567 -847.258641 -1304.332185 147.914793
 -52.523533 0.000000 0.000000 0.000000 0.000000
31 5 464.215957 26150.67 23410.00 3903.86 120.617965 -129.233534 -21.495845 136.974974
 -6.933070 0.000000 0.000000 0.000000 0.000000
30 15 3
31 15 464.215957 26150.67 23410.00 3903.86 531.071064 -847.327283 -1300.620964 147.922198
-52.444675 0.000000 0.000000 0.000000 0.000000
31 5 464.733861 26213.14 23343.07 3892.73 120.617965 -129.233534 -21.495845 136.974974
 -6.933070 0.000000 0.000000 0.000000 0.000000
30 16 3
31 16 464.733861 26213.14 23343.07 3892.73 530.980100 -847.384289 -1296.910042 147.928349
 -52.365539 0.000000 0.000000 0.000000 0.000000
31 14 465.835183 27223.42 21666.20 1127.42 531.180567 -847.258641 -1304.332185 147.914793
 -52.523533 0.000000 0.000000 0.000000 0.000000
31 14 465.931065 27274.35 21584.96 1002.35 531.180567 -847.258641 -1304.332185 147.914793
 -52.523533 0.000000 0.000000 0.000000 0.000000
31 14 466.027322 27325.48 21503.41 876.80 531.180567 -847.258641 -1304.332185 147.914793
 -52.523533 0.000000 0.000000 0.000000 0.000000
31 15 466.257631 27234.94 21680.03 1248.42 531.071064 -847.327283 -1300.620964 147.922198
 -52.444675 0.000000 0.000000 0.000000 0.000000
31 15 466.353131 27285.66 21599.11 1124.21 531.071064 -847.327283 -1300.620964 147.922198
 -52.444675 0.000000 0.000000 0.000000 0.000000
31 15 466.449004 27336.58 21517.88 999.52 531.071064 -847.327283 -1300.620964 147.922198
 -52.444675 0.000000 0.000000 0.000000 0.000000
31 16 466.680435 27246.73 21693.57 1368.20 530.980100 -847.384289 -1296.910042 147.928349
 -52.365539 0.000000 0.000000 0.000000 0.000000
31 14 466.698013 27681.74 20935.16 2.00 -658.504608 752.576695 0.000000 318.813964 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
31 14 466.741048 27653.40 20967.54 2.00 -658.504608 752.576695 0.000000 318.813964 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
31 6 466.741048 27653.40 20967.54 2.00 -9.312661 10.643042 0.000000 318.813964 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
33 14 466.841048
33 6 466.841048
31 16 466.775551 27297.23 21612.97 1244.84 530.980100 -847.384289 -1296.910042 147.928349
-52.365539 0.000000 0.000000 0.000000 0.000000
31 16 466.871038 27347.94 21532.06 1121.00 530.980100 -847.384289 -1296.910042 147.928349
-52.365539 0.000000 0.000000 0.000000 0.000000
31 15 467.215957 27743.89 20868.02 2.00 -658.966390 752.172385 0.000000 318.778798 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
31 15 467.258992 27715.52 20900.39 2.00 -658.966390 752.172385 0.000000 318.778798 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
31 7 467.258992 27715.52 20900.39 2.00 -9.319192 10.637324 0.000000 318.778798 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
33 15 467.358992
33 7 467.358992
31 16 467.733861 27806.08 20800.91 2.00 -659.419504 751.775177 0.000000 318.744273 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
31 16 467.776896 27777.70 20833.27 2.00 -659.419504 751.775177 0.000000 318.744273 0.000000
 0.000000 0.000000 0.000000 0.000000 0.000000
```

```
31 8 467.776896 27777.70 20833.27 2.00 -9.325600 10.631707 0.000000 318.744273 0.000000
 0.000000 0.000000 0.000000 0.000000
33 16 467.876896
33 8 467.876896
31 3 540.276717 37920.00 80.00 4000.00 -212.132034 0.000000 -22.368045 269.999848
 -6.019257 0.000000 0.000000 0.000000 0.000000
31 5 645.360670 48000.00 0.00 10.00 0.000000 0.000000 0.000000 136.974974 0.000000
0.000000 0.000000 0.000000 0.000000
31 3 718.656188 80.00 80.00 10.00 0.000000 0.000000 0.000000 269.999848 0.000000
 0.000000 0.000000 0.000000 0.000000
86 718.656188
```

Appendix C. SUPPORTING CODE, USERS MANUAL FOR THE GENERIC DRIVER AND LINKED LIST CODE

C.1 Generic Linked List

C.1.1 General Description This program can be used to create instances of a PRIORITY, LIFO, or FIFO queues. The header file (ll.h) contains key define statements and the prototypes of the functions available to manipulate the instantiated queues.

C.1.2 Reference Descriptions are written in the following format:

Function name

- **Summary**
- **Description**
- **Return Value**
- **Example**

Below the name of the function, the summary shows an exact syntax model for it and the Description outlines its actual effects. The return value type is given and is often useful to test for error condition if one is given before the results of the function call is used. Examples are referenced in the included code, where needed new code is included to present the example.

ll_clear

- **Summary**

#include "ll.h"

struct linked_list* ll_clear (l_list)

struct linked_list* l_list;

- **Description**

The ll_clear function allows the user to empty a l_list leaving a list with no elements.

- **Return Value**

The return value is a pointer to the list which was emptied by using this function or is NULL if the function call was not made to a valid list.

- **Example**

See function end_sim (3.3) in sim_driv.c.

ll_delete

- **Summary**

#include "ll.h"

struct linked_list* ll_delete (l_list, equal_free ...);

struct linked_list* l_list;

int (*equal_free)();

void* equal_free_arguments;

- **Description**

The ll_delete function allows the user to delete one or more occurrences of an item from l_list. The function equal_free will be called from within ll_delete. Equal_free will be passed a pointer to the data contained within an element of l_list. The equal_free_arguments value can be used as a utility pointer. however, its most common use is to pass ll_delete changing identifier values which equal_free will use in it's comparison process to determine if an item should be deleted. Thus, ll_delete knowing which list (l_list) is being referenced will match, using the function equal_free, the data in the list with the data passed as equal_free_arguments. Items matched will be removed from the list. It is equal_frees job to:

1. Deallocate the memory used to store 'data', if desired.
2. Let ll_delete know whether to;

- delete the node and stop searching for items to delete.
- delete the node and continue looking for another item to delete.
- not delete and stop.
- not delete but continue searching.

The choices are given as the return value of equal_free and are the consequence of a bitwise 'or' of the following defines from ll.h:

- LL_STOP 0x1000
- LL_CONTINUE 0x0000
- LL_DEL_YES 0x0001
- LL_DEL_NO 0x0000

- **Return Value**

The ll_delete function returns a pointer to a FIFO list containing the deleted data. If no data is deleted or if the function call was not made to a valid list NULL is returned.

- **Example**

```
#include <string.h>
#include <stdio.h>
#include "ll.h"

int equal_free (char* data_to_match);

char name [80][80];

main()
{
struct linked_list *PQ;
struct linked_list *data_list;
int(*compare)();
FILE* source;
int i = 0;
```

```
void* output;
char* name_ptr;
compare = strcmp;

PQ = ll_make (PRIORITY, compare);
data_list = ll_make (FIFO);
source = fopen (''namefile.c'', ''r'');
while ((name_ptr = fgets(name[i], 80, source)) != NULL)
{
ll_insert (PQ, name_ptr);
i++
}
fclose (source);
ll_delete (PQ, equal_free);
printf (''%d\n'', ll_length(PQ));
while ((output = ll_pop(PQ) != NULL)
printf (''%s\n'', output);
printf (''%d\n'', ll_length(PQ));
while (!(ll_isempty (data_list)))
printf ("%s\n", ll_pop (data_list));
}

int equal_free (data_from_list)
char* data_from_list;
{
int temp, result;
char* cs = ''kathy'';
int n = 5;
temp = strncmp (cs, data_from_list, n);
if (temp == 0)
result = LL_DEL_YES | LL_CONTINUE;
/* result = LL_DEL_YES | LL_STOP; */
else
result = LL_DEL_NO | LL_CONTINUE;
return result;
}
```

This program reads a set of strings from the file 'namefile.c' inserting each string into PQ based on the strcmp function. After the file has been read, items matching the string "kathy" are deleted. Either one occurence can be deleted or all occurrences can be deleted based on the return value of equal_free.

ll_insert

- **Summary**

#include "ll.h"

void* ll_insert (l_list, data)

struct linked_list* l_list;

void* data;

- **Description**

The ll_insert function allows the user to insert items into the list. How an item is inserted is dependent on what type of list has been created, either a LIFO, FIFO, or PRIORITY. Respectively, items are inserted immediately after the head, at the tail, or by some priority mechanism. The sorting methodology used when inserting into a priority queue is passed into the structure of the queue when it is instantiated (see ll_make).

- **Return Value**

The ll_insert function returns a pointer to the data which has been inserted or NULL if the function call was not made to a valid list or not enough memory space exists to accomodate the new element data.

- **Example**

See function car_arrives (1.5) in hogwash.c.

ll_isempty

- **Summary**

#include "ll.h"

int ll_isempty (l_list)

struct linked_list* l_list;

- **Description**

The ll_isempty function gives the user the ability to directly determine if the list has any elements.

- **Return Value**

The ll_isempty function returns true if the list is empty and false if it is not empty. If the function call was not made to a valid list the return value is NULL.

- **Example**

See function end_wash (1.7) in hogwash.c.

ll_length

- **Summary**

#include "ll.h"

int ll_length (l_list)

struct linked_list* l_list;

- **Description**

The ll_length functions gives the user the ability to directly determine how many items are in the list.

- **Return Value**

The ll_length function returns the integer value of the number of elements in the list or NULL if the function call is not to a valid list.

- **Example**

```
#include <string.h>
#include <stdio.h>
#include ''ll.h''

int equal_free (char* data_to_match);

char name [80][80];

main()
{
struct linked_list *PQ;
int(*compare)();
FILE* source;
int i = 0;
void* output;
```

```
char* name_ptr;
compare = strcmp;

PQ = ll_make (PRIORITY, compare);
source = fopen (''namefile.c'', ''r'');
while ((name_ptr = fgets(name[i], 80, source)) != NULL)
{
ll_insert (PQ, name_ptr);
i++
}
fclose (source);
ll_delete (PQ, equal_free);
printf (''%d\n'', ll_length(PQ));
while ((output = ll_pop(PQ) != NULL)
printf (''%s\n'', output);
printf (''%d\n'', ll_length(PQ));
}

int equal_free (data_from_list)
char* data_from_list;
{
int temp, result;
char* cs = ''kathy'';
int n = 5;
temp = strncmp (cs, data_from_list, n);
if (temp == 0)
result = LL_DEL_YES | LL_CONTINUE;
/* result = LL_DEL_YES | LL_STOP; */
else
result = LL_DEL_NO | CONTINUE;
return result;
}
```

ll_make

- **Summary**

  #include "ll.h"

  struct linked_list* ll_make (type, ...)

  int type;

  unsigned int (*compare)(); optional

  void* utility_ptr; optional

- **Description**

  The ll_make function gives the user the ability to create a queue by choosing a 'type' LIFO, FIFO, or PRIORITY. If the user chooses a PRIORITY queue then they must provide a pointer to a function which will be used by the list to determine where an item gets inserted. The optional utility pointer argument is provided to give the user added flexibility in using the functions provided by this implementation. An example of a possible use for this pointer is in back referencing which may be required when using an intermediate level where the user supplied function is not given directly to the ll_make function.

- **Return Value**

  The ll_make function returns a pointer to the newly created queue or NULL if there is not enough memory to create the queue.

- **Example**

See function main (1.0) in hogwash.c.

ll_pop

- **Summary**

#include"ll.h"

void* ll_pop (l_list)

struct linked_list* l_list;

- **Description**

The ll_pop function allows the user to take items off the top of the queue. Once an item is popped it is no longer in the queue.

- **Return Value**

The ll_pop function returns a pointer to the data which has just been popped from the queue or NULL if there are no items to be popped or the function call was not to a valid list.

- **Example**

See function end_wash (1.7) in hogwash.c.

C.1.3 The Generic Linked List Code (ll.h, ll.c)

```
/*************************************************************************/
/*****************    This is the linked list header file, ll.h    ***********/
/*************************************************************************/
#define FIFO 1
#define LIFO 2
#define PRIORITY 3
#define LL_STOP     0x1000
#define LL_CONTINUE 0x0000
#define LL_DEL_YES  0x0001
#define LL_DEL_NO   0x0000

/*--------------------------- PROTOTYPES ----------------------------------*/

struct linked_list *ll_delete (); /* struct linked_list *l_list, int (*equal_free)(),
... [equal_free_arguments] */
void *ll_insert (); /* struct linked_list *l_list, void *data */
int ll_isempty (); /* struct linked_list *l_list */
struct linked_list *ll_make (); /* int type, ...   [unsigned int(*compare)(), void *utility_ptr]  */
void *ll_pop(); /* struct linked_list *l_list */
struct linked_list *ll_clear (); /* struct linked_list *l_list */
int ll_length (); /* struct linked_list *l_list */

/*************************************************************************/



/*************************************************************************/
/*  DATE: 03/05/90                                                       */
/*  VERSION: 0.0                                                         */
/*  TITLE: Generic linked list.                                          */
/*  FILENAME: ll.c                                                       */
/*  COORDINATOR: Rob Rizza                                               */
/*  PROJECT: EENG 650, winter 90, Bisbee                                 */
/*  OPERATING SYSTEM: MS-DOS                                             */
/*  LANGUAGE: Microsoft Quick-C                                          */
/*  FILE PROCESSING: compile and link with file which uses a linked list */
/*  CONTENTS:   2.0 ll_make - used to make an instance of linked list    */
/*              2.1 ll_pop - returns top item from the list              */
/*              2.2 ll_clear - empties the linked list                   */
/*              2.3 ll_delete - deletes a selected item(s) from the list */
/*              2.4 ll_isempty - returns true if list is empty, else false */
/*              2.5 ll_length - returns the # of elements in the list    */
/*              2.6 ll_insert - inserts element into the list by calling */
/*                              ll_pinsert, ll_linsert, or ll_finsert    */
/*              2.6.1 ll_pinsert - inserts element based on a priority   */
/*              2.6.2 ll_linsert - inserts element onto the top of list  */
/*              2.6.3 ll_finsert - inserts element onto bottom of list   */
/*  FUNCTION: Allows the user the ability to create an instance of a LIFO, */
/*            FIFO, or PRIORITY queue.                                   */
/*************************************************************************/
/*                          Code begins here                             */
/*-----------------------------------------------------------------------*/
#include "ll.h"
/* #include <stdlib.h>  **** comment out to run on sun *******/
#include <stdio.h>
#include <malloc.h>

#define TRUE 1
#define FALSE 0
#define LL_MAGIC 0x12345678   /* used to check for valid pointer addressing */
```

```
/************************************ STRUCTURES ************************************/
struct list_element
{
void *data;
struct list_element *next;
};

/*--------------------------------------------------------------------------------*/

struct linked_list
{
unsigned long magic;
int typell;
struct list_element *tail;
int (*compare)(); /* void*, void*, void* */
void* utility_ptr;
struct list_element head;
};

/*********************************************************************************/
/*  DATE: 03/05/90                                                               */
/*  VERSION: 0.0                                                                 */
/*  NAME: ll_make                                                                */
/*  MODULE NUMBER: 2.0                                                           */
/*  DESCRIPTION: User can instantiate a linked list with this function           */
/*  ALGORITHM: Allocate memory for linked_list structure                         */
/*             initialize the structure                                          */
/*  PASSED VARIABLES: type, (*compare)(), *utility_ptr                           */
/*  RETURNS: struct linked_list* temp                                            */
/*  GLOBAL VARIABLES PASSED: none                                                */
/*  GLOBAL VARIABLES CHANGED: none                                               */
/*  FILES READ: none                                                             */
/*  FILES WRITTEN: none                                                          */
/*  HARDWARE INPUT: none                                                         */
/*  HARDWARE OUTPUT: none                                                        */
/*  MODULES CALLED: none                                                         */
/*  CALLING MODULES: whatever executable file is using the data structure        */
/*  ORDER OF: This function is of order O(1)                                     */
/*  AUTHOR: Capt Rob Rizza                                                       */
/*  HISTORY: none                                                                */
/*********************************************************************************/
struct linked_list* ll_make(type, compare, utility_ptr)
int type;
int (*compare)();
void* utility_ptr;
{
   struct linked_list *temp = NULL;

   if((temp=(struct linked_list*) malloc(sizeof (struct linked_list)))==NULL)
      return(temp);

   temp->typell=type;
   temp->head.next=NULL;
   temp->tail=NULL;
   temp->magic=LL_MAGIC;
   temp->utility_ptr=utility_ptr;

   if(type==PRIORITY)
      temp->compare=compare;

   else
      temp->compare=NULL;

   return(temp);
}
```

```
/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: ll_pop                                                             */
/*  MODULE NUMBER: 2.1                                                       */
/*  DESCRIPTION: Returns the top item from the linked list                   */
/*  ALGORITHM: Remove the data from the top of the list                      */
/*             save pointer                                                  */
/*             adjust pointer to new top element                             */
/*             deallocate saved pointer                                      */
/*             return data                                                   */
/*  PASSED VARIABLES: struct linked_list* l_list                             */
/*  RETURNS: *ll_data                                                        */
/*  GLOBAL VARIABLES PASSED: none                                            */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: none                                                     */
/*  CALLING MODULES: whatever executable file is using the data structure    */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
void *ll_pop(l_list)
struct linked_list *l_list;
{
void *ll_data  = NULL;
struct list_element *temp = NULL;

if (l_list->magic != LL_MAGIC)
{
printf ("Magic number test failed in ll_pop, check pointer\n");
return NULL;
}
if (l_list->head.next != NULL)
ll_data = l_list->head.next->data;
else
{
printf ("Cannot pop from empty list\n");
return NULL;
}
temp = l_list->head.next;
l_list->head.next = l_list->head.next->next;

if (l_list->head.next==NULL)
l_list->tail=NULL;

free(temp);

return ll_data;
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: ll_clear                                                           */
/*  MODULE NUMBER: 2.2                                                       */
/*  DESCRIPTION: Empties the linked list                                     */
/*  ALGORITHM: While there are elements in the list                          */
/*                      pop items off                                        */
/*                      deallocate memory                                    */
/*             return pointer to the empty list                              */
```

```
/*  PASSED VARIABLES: struct linked_list* l_list                              */
/*  RETURNS: struct linked_list* l_list                                       */
/*  GLOBAL VARIABLES PASSED: none                                             */
/*  GLOBAL VARIABLES CHANGED: none                                            */
/*  FILES READ: none                                                          */
/*  FILES WRITTEN: none                                                       */
/*  HARDWARE INPUT: none                                                      */
/*  HARDWARE OUTPUT: none                                                     */
/*  MODULES CALLED: none                                                      */
/*  CALLING MODULES: whatever executable file is using the data structure     */
/*  ORDER OF: This function is of order O(n) where n = #items in the list     */
/*  AUTHOR: Capt Rob Rizza                                                    */
/*  HISTORY: none                                                             */
/****************************************************************************/
struct linked_list *ll_clear (l_list)
struct linked_list *l_list;
{
 struct list_element *node = NULL, *temp = NULL;

 if (l_list->magic != LL_MAGIC)
 {
printf ("Magic number test failed in ll_clear, check pointer\n");
return NULL;
 }

 node = &l_list->head;

 while (node->next != NULL)
 {
temp = node->next->next;
free (node->next->data);
free (node->next);
node->next = temp;
 }
 return l_list;
}

/* ************************************************************************/
/*  DATE: 03/05/90                                                        */
/*  VERSION: 0.0                                                          */
/*  NAME: ll_delete                                                       */
/*  MODULE NUMBER: 2.3                                                    */
/*  DESCRIPTION: Deletes a selected item(s) from the list                 */
/*  ALGORITHM: While there are elements in the list                       */
/*                     match item to be deleted                           */
/*                     delete and stop search for matching items or       */
/*                     delete and continue looking for items to delete    */
/*                     return a ptr to the last item deleted              */
/*  PASSED VARIABLES: struct linked_list* l_list, (*equal_free)()         */
/*  RETURNS: ptr_to_data                                                  */
/*  GLOBAL VARIABLES PASSED: none                                         */
/*  GLOBAL VARIABLES CHANGED: none                                        */
/*  FILES READ: none                                                      */
/*  FILES WRITTEN: none                                                   */
/*  HARDWARE INPUT: none                                                  */
/*  HARDWARE OUTPUT: none                                                 */
/*  MODULES CALLED: none                                                  */
/*  CALLING MODULES: whatever executable file is using the data structure */
/*  ORDER OF: This function is of order O(n) where n = #items in the list */
/*  AUTHOR: Capt Rob Rizza                                                */
/*  HISTORY: none                                                         */
/****************************************************************************/
struct linked_list *ll_delete(l_list, equal_free, equal_free_arguments)
struct linked_list *l_list;
int (*equal_free)();
```

```
void *equal_free_arguments;
{
unsigned int result;
void *ptr_to_data = NULL;
struct list_element *node = NULL, *temp = NULL;
struct linked_list *deleted_data_list = NULL;

deleted_data_list = ll_make (FIFO);

if (l_list->magic != LL_MAGIC)
{
       printf ("Magic number test failed in ll_delete, check pointer\n");
       return NULL;
}

node = &l_list->head;

while (node->next != NULL)
{
result = (*equal_free)(node->next->data, equal_free_arguments);

if(result & LL_DEL_YES)  /* if result has a 1 in the LSB position */
{                        /* then delete                           */
 temp = node->next;
 ll_insert (deleted_data_list, node->next->data);
 node->next = node->next->next;

 if (node->next == NULL)
l_list->tail = node;

 free(temp);
}
if(result & LL_STOP)     /* if result has a 1 in the MSB position */
break;          /* then stop                             */

if (!(result & LL_DEL_YES))  /* if result has a 0 in the MSB   */
node = node->next;           /* position then continue         */
}
return deleted_data_list;
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: ll_isempty                                                         */
/*  MODULE NUMBER: 2.4                                                       */
/*  DESCRIPTION: Returns true if the list is empty, else return false        */
/*  ALGORITHM: If there is an element in the list return true,               */
/*  PASSED VARIABLES: struct linked_list* l_list                             */
/*  RETURNS: true or false                                                   */
/*  GLOBAL VARIABLES PASSED: none                                            */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: none                                                     */
/*  CALLING MODULES: whatever executable file is using the data structure    */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
int ll_isempty(l_list)
struct linked_list *l_list;
{
```

```
if (l_list->magic != LL_MAGIC)
{
       printf ("Magic number test failed in ll_isempty, check pointer\n");
       return NULL;
}

if (l_list->head.next == NULL)
return TRUE;

return FALSE;
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: ll_length                                                          */
/*  MODULE NUMBER: 2.5                                                       */
/*  DESCRIPTION: Returns the number of elements in the list                  */
/*  ALGORITHM: While there is an element in the list                         */
/*                       increment the counter                               */
/*             return counter                                                */
/*  PASSED VARIABLES: struct linked_list* l_list                             */
/*  RETURNS: i (# of elements in the list)                                   */
/*  GLOBAL VARIABLES PASSED: none                                            */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: none                                                     */
/*  CALLING MODULES: whatever executable file is using the data structure    */
/*  ORDER OF: This function is of order O(n) where n = #items in the list    */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
int ll_length (l_list)
struct linked_list *l_list;
{
int i = 0;
struct list_element *node = NULL;

if (l_list->magic != LL_MAGIC)
{
       printf ("Magic number test failed in ll_length, check pointer\n");
       return NULL;
}

node = &l_list->head;

while (node->next != NULL)
{
i++;
node = node->next;
}
return i;
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: ll_insert                                                          */
/*  MODULE NUMBER: 2.6                                                       */
/*  DESCRIPTION: Inserts an element into the list by calling ll_pinsert,     */
/*               ll_linsert, or ll_finsert                                   */
/*  ALGORITHM: switch to selected type                                       */
```

```
/*  PASSED VARIABLES: l_list, data                                          */
/*  RETURNS: data                                                           */
/*  GLOBAL VARIABLES PASSED: none                                           */
/*  GLOBAL VARIABLES CHANGED: none                                          */
/*  FILES READ: none                                                        */
/*  FILES WRITTEN: none                                                     */
/*  HARDWARE INPUT: none                                                    */
/*  HARDWARE OUTPUT: none                                                   */
/*  MODULES CALLED: ll_pinsert (2.6.1), ll_linsert (2.6.2), or              */
/*                  ll_finsert (2.6.3)                                      */
/*  CALLING MODULES: whatever executable file is using the data structure   */
/*  ORDER OF: This function is of order O(1)                                */
/*  AUTHOR: Capt Rob Rizza                                                  */
/*  HISTORY: none                                                           */
/*************************************************************************/
static void ll_linsert (); /* struct linked_list *l_list, struct list_element *new_element */
static void ll_pinsert (); /* struct linked_list *l_list, struct list_element *new_element */
static void ll_finsert (); /* struct linked_list *l_list, struct list_element *new_element */

void * ll_insert(l_list,data)
struct linked_list *l_list;
void *data;

{
struct list_element *new_element = NULL;

if(l_list->magic!=LL_MAGIC)
{
       printf ("Magic number test failed in ll_insert, check pointer\n");
       return NULL;
}

if((new_element=(struct list_element *) malloc(sizeof( struct list_element)))==NULL)
return(new_element);

new_element->data=data;

switch(l_list->typell)
{
case PRIORITY: ll_pinsert(l_list, new_element);
                    break;
case LIFO    : ll_linsert(l_list, new_element);
                    break;
case FIFO    : ll_finsert(l_list, new_element);
                    break;
default      : return(NULL);
}
return(new_element->data);
}

/*************************************************************************/
/*  DATE: 03/05/90                                                         */
/*  VERSION: 0.0                                                           */
/*  NAME: ll_pinsert                                                       */
/*  MODULE NUMBER: 2.6.1                                                   */
/*  DESCRIPTION: Inserts an element into the list based on a user supplied */
/*               comparison function                                       */
/*  ALGORITHM: While there are items in the list                           */
/*                     compare old and new item using compare function     */
/*                     if new <= old continue comparing                    */
/*                     else insert into list                               */
/*  PASSED VARIABLES: struct linked_list* l_list, struct list_element*     */
/*                    new_element                                          */
/*  RETURNS: none                                                          */
/*  GLOBAL VARIABLES PASSED: none                                          */
```

```
/*  GLOBAL VARIABLES CHANGED: none                                          */
/*  FILES READ: none                                                        */
/*  FILES WRITTEN: none                                                     */
/*  HARDWARE INPUT: none                                                    */
/*  HARDWARE OUTPUT: none                                                   */
/*  MODULES CALLED: none                                                    */
/*  CALLING MODULES: ll_insert (2.6)                                        */
/*  ORDER OF: This function is of order O(n) where n = #items in the list   */
/*  AUTHOR: Capt Rob Rizza                                                  */
/*  HISTORY: none                                                           */
/**************************************************************************/
static void ll_pinsert(l_list, new_element)
struct linked_list* l_list;
struct list_element* new_element;
{
struct list_element *node = NULL,*temp = NULL;

node=&l_list->head;

while(node->next != NULL && (*(l_list->compare))(new_element->data, node->next->data, l_list->utility_ptr) <= 0)
node=node->next;    /* loop to find where the item will be inserted */
    /* based on user defined compare function       */
temp=node->next;
node->next=new_element;
new_element->next=temp;

if (new_element->next == NULL)       /* reset tail ptr if new_element is the tail */
l_list->tail = new_element;
}

/**************************************************************************/
/*  DATE: 03/05/90                                                          */
/*  VERSION: 0.0                                                            */
/*  NAME: ll_linsert                                                        */
/*  MODULE NUMBER: 2.6.2                                                    */
/*  DESCRIPTION: Inserts an element at the top of the list                  */
/*  ALGORITHM: Insert element at the top of the list                        */
/*             adjust pointers                                              */
/*  PASSED VARIABLES: struct linked_list* l_list, struct list_element*      */
/*                    new_element                                           */
/*  RETURNS: none                                                           */
/*  GLOBAL VARIABLES PASSED: none                                           */
/*  GLOBAL VARIABLES CHANGED: none                                          */
/*  FILES READ: none                                                        */
/*  FILES WRITTEN: none                                                     */
/*  HARDWARE INPUT: none                                                    */
/*  HARDWARE OUTPUT: none                                                   */
/*  MODULES CALLED: none                                                    */
/*  CALLING MODULES: ll_insert (2.6)                                        */
/*  ORDER OF: This function is of order O(1)                                */
/*  AUTHOR: Capt Rob Rizza                                                  */
/*  HISTORY: none                                                           */
/**************************************************************************/
static void ll_linsert (l_list, new_element)   /* inserts at the head of list */
struct linked_list *l_list;
struct list_element *new_element;
{
struct list_element *node = NULL, *temp = NULL;

temp = l_list->head.next;
l_list->head.next = new_element;
new_element->next = temp;

if (new_element->next == NULL)
l_list->tail = new_element;
```

```
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: ll_finsert                                                         */
/*  MODULE NUMBER: 2.6.3                                                     */
/*  DESCRIPTION: Inserts an element at the bottom of the list                */
/*  ALGORITHM: Insert element at the bottom of the list                      */
/*             adjust pointers                                               */
/*  PASSED VARIABLES: struct linked_list* l_list, struct list_element*       */
/*                    new_element                                            */
/*  RETURNS: none                                                            */
/*  GLOBAL VARIABLES PASSED: none                                            */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: none                                                     */
/*  CALLING MODULES: ll_insert (2.6)                                         */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
static void ll_finsert(l_list, new_element)   /* inserts items at the tail of list */
struct linked_list *l_list;
struct list_element *new_element;
{
struct list_element *node = NULL;

if (l_list->tail != NULL)
{
l_list->tail->next = new_element;
new_element->next = NULL;
l_list->tail = new_element;
}
else
{
l_list->head.next = new_element;
new_element->next = NULL;
l_list->tail = new_element;
}
}
```

C.2 Generic Simulation Driver

C.2.1 General Description This program can be used to create an instance of a simulation driver. The header file (sim_driv.h) contains the functions which can be used to build and execute an event driven simulation.

C.2.2 Reference Descriptions are written in the following format:

Function name

- **Summary**
- **Description**
- **Return Value**
- **Example**

Below the name of the function, the summary shows an exact syntax model for it and the Description outlines its actual effects. The return value type is given and is often useful to test for error condition if one is given before the results of the function call is used. Examples are referenced in the included code, where needed new code is included to present the example.

delete_event

- **Summary**

#include "sim_driv.h"

struct linked_list* delete_event (driver, event_id)

struct driver* driver;

int event_id;

- **Description**

The delete_event function gives the user the ability to remove previously scheduled events from the Next Event Queue (NEQ). Using the event_id, returned to the user when using "schedule_event", delete_event searches for a matching event_id in the NEQ and deletes it.

- **Return Value**

The delete_event function returns a pointer to a structure containing the following information: *time, *event, *event_arguments, and an event_id. (see "sim_driv.h" for structure)

- **Example**

Replace the function "end_wash" in hogwash.c with the version given here. This will cause all rewashes to be unscheduled (ie. deletes all rewash events).

```
void end_wash (argument)
struct argument_list* argument;
{
struct driver_data* deleted_data;
```

```
int event_id;
        struct argument_list* temp_argument;
        unsigned int j = 0;

        printf ("CAR %d", argument->car_id);
        printf (" WASH IS FINISHED. TIME STAMP= %d\n", *argument->time);
        while (j++ < 65000)                 /* time to read screen loop */
                ;
        if (rand() % 5 == 3)                /* random selection of rewashes */
{
                event_id = schedule_event (argument->carwash, argument->time, rewash, argument);
printf ("REWASH_ID is %d\n", event_id);
deleted_data = delete_event (argument->carwash, event_id);
printf ("event_id deleted is %d\n", deleted_data->event_id);
}

        else if (!ll_isempty(line))         /* if you don't get a rewash get next start_wash */
        {                                   /* from the line if there's someone in it        */
                temp_argument = (ll_pop(line));
                argument->car_id = temp_argument->car_id;
                schedule_event (argument->carwash, argument->time, start_wash, argument);
        }
}
```

end_sim

- **Summary**

#include "sim_driv.h"

struct driver* end_sim (driver)

struct driver* driver;

- **Description**

The end_sim function gives the user the ability to stop the simulation. End_sim effectively empties the Next Event Queue (NEQ). The execute_sim function checks for an empty NEQ and terminates execution when the NEQ is empty (see the execute_sim function).

- **Return Value**

The end_sim functon returns a pointer to the simulation driver.

- **Example**

See function close_wash (1.3) in hogwash.c.

execute_sim

- **Summary**

  #include "sim_driv.h"

  struct linked_list* execute_sim (driver)

  struct driver* driver;

- **Description**

  The execute_sim function executes the functions(events) which have been scheduled with the schedule_event function. Execute_sim will continue dispatching events until there are no more events scheduled in the Next Event Queue.

- **Return Value**

  The execute_sim function returns a pointer to a FIFO queue containing the event_id and time of event for each executed event.

- **Example**

  See function main (1.0) in hogwash.c.

make_driver

- **Summary**

#include "sim_driv.h"

struct driver* make_driver (sizeof_time, compare_time)

int sizeof_time;

int (*compare_time)();

- **Description**

The make_driver function allows the user to create an instance of the simulation driver. The user can then use the functions available to manipulate the simulation driver in creating a working simulation. The compare function is supplied by the user to allow for sorting of the events.

- **Return Value**

The make_driver function return a pointer to the just created simulation driver.

- **Example**

See function main (1.0) in hogwash.c.

print_stats

- **Summary**

#include "sim_driv.h"

void print_stats (stats)

struct linked_list* stats;

- **Description**

The print_stats function provides the user with a viewable output from the returned value of execute_sim. The output is the event_id and time of event for each executed event.

- **Return Value**

Print_stats has no return value.

- **Example**

See function main (1.0) in hogwash.c

schedule_event

- **Summary**

#include "sim_driv.h"

int schedule_event (driver, time, event_func, event_func_arguments)

struct driver* driver;

void* time;

void (*event_func)();

void* event_func_arguments;

- **Description**

The schedule_event function allows the user to schedule events by passing a pointer to the event function, its arguments, and the time of the event with the simulation identifier 'driver'.

- **Return Value**

Schedule_event returns a unique event_id for each newly scheduled event. It returns NULL if the user tries to schedule an event at a time which has already passed or there is not enough memory space to schedule the event.

- **Example**

See function main (1.0) in hogwash.c.

C.2.3 The Generic simulation Driver Code (sim_driv.h, sim_driv.c)

```
/*************************************************************************/
/*************************************************************************/
/*                This is the header file for sim_driv.c                 */
/*************************************************************************/
#define LL_DEL_YES 0x0001
#define LL_DEL_NO 0x0000

struct driver *make_driver (); /*int sizeof_time, int(*compare_time)() */
int schedule_event (); /* struct driver *driver, void* time, void(*event_func)(),...   [void *event_func_arguments] */
struct linked_list *execute_sim(); /*struct driver *driver */
void print_stats (); /* struct linked_list* stats */
struct driver *end_sim (); /* struct driver* driver */
struct linked_list *delete_event (); /* struct driver* driver, int event_ident */

struct driver_data {
void *time;
void (*func)();
void *func_arguments;
int event_id;
};

/*************************************************************************/

/*************************************************************************/
/*  DATE: 03/05/90                                                       */
/*  VERSION: 0.0                                                         */
/*  TITLE: Generic simulation driver                                     */
/*  FILENAME: sim_driv.c                                                 */
/*  COORDINATOR: Rob Rizza                                               */
/*  PROJECT: EENG 650, winter 90, Bisbee                                 */
/*  OPERATING SYSTEM: MS-DOS                                             */
/*  LANGUAGE: Microsoft Quick-C                                          */
/*  FILE PROCESSING: Link and compile with ll.c and executable file which */
/*                   uses this file                                      */
/*  CONTENTS: 3.0 make_driver - allows user the ability to make an instance */
/*                              of sim_driv                              */
/*            3.1 schedule_event - allows user the ability to schedule events */
/*            3.2 execute_sim - executes the scheduled events          */
/*            3.3 end_sim - terminates the simulation                  */
/*            3.4 print_stats - prints out the event_id and time of that */
/*                              event for all executed events          */
/*  FUNCTION: Gives a user the basic functions needed to run an event driven */
/*            simulation                                               */
/*************************************************************************/
/*                          Code begins here                             */
/*-----------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "sim_driv.h"
#include "ll.h"
#include "sim_stru.h"     /* not part of the original generic sim_driv.c code */

/*************************************************************************/
/*                    Structures used in sim_driv                        */
/*************************************************************************/

struct driver {
int sizeof_time;
struct driver_data *curr_event;
struct linked_list *REQ;
unsigned long event_id;
```

```
int (*compare)(); /* void*, void* */
};

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: Make driver                                                        */
/*  MODULE NUMBER: 3.0                                                       */
/*  DESCRIPTION: Allows the user to create an instance of sim_driv           */
/*  ALGORITHM: Allocate memory for struct driver                             */
/*             make an instance of priority queue                            */
/*             load struct driver                                            */
/*             return driver                                                 */
/*  PASSED VARIABLES: sizeof_time, (*compare_time)()                         */
/*  RETURNS: driver                                                          */
/*  GLOBAL VARIABLES USED: none                                              */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: ll_make (2.0)                                            */
/*  CALLING MODULES: main (1.0)                                              */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
int new_compare_time (); /* struct driver_data* new_data, struct driver_data* old_data, struct driver* driver */

struct driver *make_driver (sizeof_time, compare_time)
int sizeof_time;
int (*compare_time)();
{
struct driver *driver = NULL;
struct linked_list *NEQ = NULL;

if ((driver = (struct driver*)malloc(sizeof(struct driver)))==NULL)
return driver;

NEQ = ll_make (PRIORITY, new_compare_time, driver);       /* creating the Next Event Queue (NEQ) */

driver->curr_event = NULL;
driver->NEQ = NEQ;
driver->event_id = 0;
driver->sizeof_time = sizeof_time;
driver->compare = compare_time;

return driver;
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: Compare time function                                              */
/*  MODULE NUMBER: 3.0.1                                                     */
/*  DESCRIPTION: This is the function that is passed to linked list          */
/*  ALGORITHM: Unpack and pass to compare                                    */
/*  PASSED VARIABLES: *new_data, *old_data, *driver                          */
/*  RETURNS: answer                                                          */
/*  GLOBAL VARIABLES USED: none                                              */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
```

```
/*  MODULES CALLED: compare_time (1.1)                                          */
/*  CALLING MODULES: ll_pinsert (2.6.1)                                        */
/*  ORDER OF: This function is of order O(1)                                   */
/*  AUTHOR: Capt Rob Rizza                                                     */
/*  HISTORY: none                                                              */
/*****************************************************************************/
int new_compare_time (new_data, old_data, driver)
struct driver_data *new_data;
struct driver_data *old_data;
struct driver *driver;
{
int answer;

answer = driver->compare (new_data->time, old_data->time); /* the time parts are extracted from */
return (int)answer;                                        /* new_data and old_data, passed to  */
}                                                          /* the user defined compare func by  */
   /* use of a ptr to driver which has  */
   /* a ptr to the compare function     */
/*****************************************************************************/
/*  DATE: 03/05/90                                                             */
/*  VERSION: 0.0                                                               */
/*  NAME: Schedule event function                                              */
/*  MODULE NUMBER: 3.1                                                         */
/*  DESCRIPTION: Allows the user to schedule events in the simulation          */
/*  ALGORITHM: Allocate memory for struct driver_data                          */
/*             check to see if event can be scheduled                          */
/*             load struct driver_data                                         */
/*             insert driver_data into NEQ                                     */
/*             return event_id                                                 */
/*  PASSED VARIABLES: *driver, *time, (*event_func)(), *event_func_arguments   */
/*  RETURNS: event_id                                                          */
/*  GLOBAL VARIABLES USED: none                                                */
/*  GLOBAL VARIABLES CHANGED: none                                             */
/*  FILES READ: none                                                           */
/*  FILES WRITTEN: none                                                        */
/*  HARDWARE INPUT: none                                                       */
/*  HARDWARE OUTPUT: none                                                      */
/*  MODULES CALLED: ll_insert                                                  */
/*  CALLING MODULES: main (1.0), open_wash (1.4), start_wash (1.6),            */
/*                   end_wash (1.7, rewash (1.8)                               */
/*  ORDER OF: This function is of order O(1)                                   */
/*  AUTHOR: Capt Rob Rizza                                                     */
/*  HISTORY: none                                                              */
/*****************************************************************************/
int schedule_event (driver, time, event_func, event_func_arguments)
struct driver *driver;
void *time;
void (*event_func)();
void *event_func_arguments;
{
int i = 0;
struct driver_data *new_event_data = NULL;

if ((new_event_data = (struct driver_data*)malloc(sizeof(struct driver_data)))==NULL)
return NULL;

if (driver->curr_event != NULL)
if (driver->compare (driver->curr_event->time, time) < 0)  /* (*time < *(driver->curr_event->time)) */
{
printf ("Cannot schedule event, new event time is already history\n");
return NULL;
}

new_event_data->func = event_func;
new_event_data->func_arguments = event_func_arguments;
```

```
new_event_data->event_id = (int)((driver->event_id = (++driver->event_id)));
new_event_data->time = (void*)malloc(sizeof(double));
new_event_data->time = (double*)memcpy(new_event_data->time, time, driver->sizeof_time);
      /* (double*) cast was not in original sim_driv.c code */
ll_insert(driver->NEQ, new_event_data);

return new_event_data->event_id;
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: Execute Simulation function                                        */
/*  MODULE NUMBER: 3.2                                                       */
/*  DESCRIPTION: Allows the user to execute the simulation                   */
/*  ALGORITHM: While there are events scheduled                              */
/*                      execute the event                                    */
/*                      load stats queue                                     */
/*            return sim_stats                                               */
/*  PASSED VARIABLES: *driver                                                */
/*  RETURNS: struct linked_list* sim_stats                                   */
/*  GLOBAL VARIABLES USED: none                                              */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: ll_make (2.0), ll_pop (2.1), ll_insert (2.6)             */
/*  CALLING MODULES: main (1.0)                                              */
/*  ORDER OF: This function is of order O(n) where n = #events in NEQ        */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
struct linked_list *execute_sim(driver)
struct driver *driver;
{
struct driver_data *sim_info = NULL;
struct linked_list *sim_stats = NULL;

sim_stats = ll_make(LIFO);       /* creating the stats queue */

while (!(ll_isempty(driver->NEQ)))
      {
      sim_info = ll_pop (driver->NEQ),
      driver->curr_event = sim_info;  /* This allows driver to keep track of the current event */
      (*(sim_info->func))(sim_info->func_arguments);  /* execute function (event) popped from NEQ */
      ll_insert(sim_stats, sim_info);   /* putting the sim_info into the stats queue */
      }

return sim_stats;
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: End Simulation function                                            */
/*  MODULE NUMBER: 3.3                                                       */
/*  DESCRIPTION: Allows the user to end the simulation                       */
/*  ALGORITHM: empty the NEQ                                                 */
/*  PASSED VARIABLES: struct driver* driver                                  */
/*  RETURNS: ll_clear (driver->NEQ)                                          */
/*  GLOBAL VARIABLES USED: none                                              */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
```

```
/*  HARDWARE INPUT: none                                                    */
/*  HARDWARE OUTPUT: none                                                   */
/*  MODULES CALLED: ll_clear (2.2)                                          */
/*  CALLING MODULES: close_wash (1.3)                                       */
/*  ORDER OF: This function is of order O(1)                                */
/*  AUTHOR: Capt Rob Rizza                                                  */
/*  HISTORY: none                                                           */
/*************************************************************************/
struct driver *end_sim (driver)
struct driver *driver;
{

return (struct driver*)ll_clear (driver->NEQ);
}

/*************************************************************************/
/*  DATE: 03/05/90                                                         */
/*  VERSION: 0.0                                                           */
/*  NAME: Print stats function                                             */
/*  MODULE NUMBER: 3.4                                                     */
/*  DESCRIPTION: Allows the user to see the event_id and it's time for every */
/*               event                                                     */
/*  ALGORITHM: While output queue is not empty                             */
/*                     print event_id and it's time                        */
/*  PASSED VARIABLES: struct linked_list* stats                            */
/*  RETURNS: none                                                          */
/*  GLOBAL VARIABLES USED: none                                            */
/*  GLOBAL VARIABLES CHANGED: none                                         */
/*  FILES READ: none                                                       */
/*  FILES WRITTEN: none                                                    */
/*  HARDWARE INPUT: none                                                   */
/*  HARDWARE OUTPUT: none                                                  */
/*  MODULES CALLED: ll_pop                                                 */
/*  CALLING MODULES: none                                                  */
/*  ORDER OF: This function is of order O(n) where n = #events in stats queue */
/*  AUTHOR: Capt Rob Rizza                                                 */
/*  HISTORY: none                                                          */
/*************************************************************************/
void print_stats (stats)
struct linked_list *stats;
{
struct driver_data *output = NULL;

while ((output = (struct driver_data*)ll_pop(stats)) != NULL)
{
int time;
printf ("%d\t", output->event_id);
time = *(int*)output->time;
printf ("%d\n", time);
}
}

/*************************************************************************/
/*  DATE: 07/07/90                                                         */
/*  VERSION: 0.0                                                           */
/*  NAME: delete event function                                            */
/*  MODULE NUMBER: 3.5                                                     */
/*  DESCRIPTION: Allows the user to delete previously scheduled events using */
/*               event_id as the deletion reference                        */
/*  ALGORITHM: While next->item in NEQ queue is not NULL                   */
/*                     delete items with the referenced event_id           */
/*  PASSED VARIABLES: struct driver* driver, int event_id                  */
/*  RETURNS: struct linked_list*                                           */
/*  GLOBAL VARIABLES USED:                                                 */
/*  GLOBAL VARIABLES CHANGED: none                                         */
```

```
/*  FILES READ: none                                                        */
/*  FILES WRITTEN: none                                                     */
/*  HARDWARE INPUT: none                                                    */
/*  HARDWARE OUTPUT: none                                                   */
/*  MODULES CALLED: ll_delete                                               */
/*  CALLING MODULES: none                                                   */
/*  ORDER OF: This function is of order O(n) where n = #events in the NEQ   */
/*  AUTHOR: Capt Rob Rizza                                                  */
/*  HISTORY: none                                                           */
/**************************************************************************/

/* int equal_free (struct driver_data* event_data, int* event_id);   ** generic sim_driv.c version **

struct linked_list *delete_event (driver, event_id)
struct driver *driver;
int event_id;

{
struct driver_data *data = NULL;
struct linked_list *deleted_data = NULL;

deleted_data = ll_delete (driver->NEQ, equal_free, &event_id);
data = ll_pop (deleted_data);

return data;
}                               */


int equal_free (); /* struct driver_data* event_data, int* event_id  version used with rizsim.c */

struct linked_list *delete_event (driver, object_id)
struct driver *driver;
int object_id;

{
struct linked_list *deleted_data = NULL;

deleted_data = ll_delete (driver->NEQ, equal_free, &object_id);

return deleted_data;
}


/**************************************************************************/
/*  DATE: 07/07/90                                                        */
/*  VERSION: 0.0                                                          */
/*  NAME: Equal free function                                             */
/*  MODULE NUMBER: 3.6                                                    */
/*  DESCRIPTION: This is the generic driver's function which tells ll_delete */
/*               how it is to locate and delete items from the list       */
/*  ALGORITHM: if event_id from the item in the list == event_id referenced */
/*                     delete items with the referenced event_id          */
/*             else                                                       */
/*                  continue looking to match event_id                    */
/*  PASSED VARIABLES: struct driver_data*, int event_id                   */
/*  RETURNS: int result                                                   */
/*  GLOBAL VARIABLES USED: none                                           */
/*  GLOBAL VARIABLES CHANGED: none                                        */
/*  FILES READ: none                                                      */
/*  FILES WRITTEN: none                                                   */
/*  HARDWARE INPUT: none                                                  */
/*  HARDWARE OUTPUT: none                                                 */
/*  MODULES CALLED: ll_delete                                             */
/*  CALLING MODULES: none                                                 */
/*  ORDER OF: This function is of order O(1) since it simply does 1 comparison*/
```

```
/*  AUTHOR: Capt Rob Rizza                                                    */
/*  HISTORY: none                                                             */
/*****************************************************************************/

/* int equal_free (event_data, event_id)    ** generic sim_driv.c version **
struct driver_data *event_data;
int *event_id;
{
int result;

if (event_data->event_id == *event_id)
result = LL_DEL_YES | LL_CONTINUE;
else
result = LL_DEL_NO | LL_CONTINUE;

return result;
}                              */

int equal_free (event_data, object_id)        /* version used with rizsim.c */
struct driver_data *event_data;
int *object_id;
{
int result;
struct event_args *event_argument = NULL;

event_argument = event_data->func_arguments;

if (event_argument->object2 != NULL)
{
if ((event_argument->object1->object_id == *object_id)||
        (event_argument->object2->object_id == *object_id))

result = LL_DEL_YES;

else
result = LL_DEL_NO;
}
else
{
if (event_argument->object1->object_id == *object_id)

result = LL_DEL_YES;
else
result = LL_DEL_NO;
}
return result;
}
```

C.3 The Carwash Simulation

C.3.1 **General Description** The carwash simulation uses the functions available in the generic simulation driver to create a running event driven simulation which uses the following algorithm.

The main procedure schedules only two events, open the wash and close the wash. Open the wash schedules the first car arrival. The car arrival event checks to see how to handle each new arrival either putting them in line if the wash is busy or scheduling them for an immediate start wash if the wash is empty. The car arrival event also schedules the next car arrival event. The start wash event schedules an end wash event. The end wash event schedules selected cars for a rewash. Rewashes are done immediately. End wash schedules a start wash of the next car in line if a rewash is not scheduled. The simulation ends when close wash is executed.

C.3.2 The Carwash Simulation Code (hogwash.c)

```
/*************************************************************************/
/*  DATE: 03/05/90                                                       */
/*  VERSION: 0.0                                                         */
/*  TITLE: Carwash Simulation                                            */
/*  FILNAME: hogwash.c                                                   */
/*  COORDINATOR: Rob Rizza                                               */
/*  PROJECT: EENG 650, Winter 90, Bisbee                                 */
/*  OPERATING SYSTEM: MS-DOS                                             */
/*  LANGUAGE: Microsoft Quick-C                                          */
/*  FILE PROCESSING: Compile and link with ll.c and sim_driv.c           */
/*  CONTENTS:   1.0 main - schedules events, executes simulation        */
/*              1.1 compare_time - used to sort events                   */
/*              1.2 make_car_id - generates a new car id                 */
/*              1.3 close_wash - signals carwash is closed, ends simulation */
/*              1.4 open_wash - opens wash, generates 1st car arrival    */
/*              1.5 car_arrives - may schedule a start_wash, schedules next */
/*                                arrival                                */
/*              1.6 start_wash - schedules an end_wash                   */
/*              1.7 end_wash - may schedule a rewash or start_wash       */
/*              1.8 rewash - schedules an end_wash                       */
/*  FUNCTION: This file implements a carwash simulation. The carwash opens, */
/*            cars arrive, they either enter the wash or get in line     */
/*            After being washed some are rewashed. Simulation ends when */
/*            no more unexecuted events exist or the carwash is closed.  */
/*************************************************************************/
/************************  CODE BEGINS HERE  *****************************/
#include "ll.h"
#include "sim_driv.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
```

```
/*****************************************************************************/
/* user defined structure containing the arguments *carwash, *time, car_id   */
/*---------------------------------------------------------------------------*/
struct argument_list {
        struct driver* carwash;
        int* time;
        int car_id;
        };

/*****************************************************************************/
/* PROTOTYPES of user defined functions                                      */
/*---------------------------------------------------------------------------*/
int compare_time (int* time1, int* time2);
void close_wash (struct driver* carwash);
void open_wash (struct argument_list* argument);
void* car_arrives (struct argument_list* argument);
void start_wash (struct argument_list* argument);
void end_wash (struct argument_list* argument);
void rewash (struct argument_list* argument);

/*****************************************************************************/
/*               Global variables: line, and in_use_flag.                    */
/*                                                                           */
/*                  line - is created in main()                              */
/*                       - inserts into line occur in car_arrives()          */
/*                       - pops occur in end_wash()                          */
/*           in_use_flag - is set in start_wash() and rewash()               */
/*                       - it is checked in car_arrives()                    */
/*---------------------------------------------------------------------------*/
static struct linked_list* line;
static int in_use_flag = 0;

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: main                                                               */
/*  MODULE NUMBER: 1.0                                                       */
/*  DESCRIPTION: Creates an instance of the simulation, schedules events, and */
/*               executes the simulation.                                    */
/*  ALGORITHM: create driver - schedule events - execute events              */
/*  PASSED VARIABLES: none                                                   */
/*  RETURNS: none                                                            */
/*  GLOBAL VARIABLES USED: line                                              */
/*  GLOBAL VARIABLES CHANGED: line is created                                */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none, except by redirection at run-time                   */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: make_driver (3.0), schedule_event (3.1),                 */
/*                  ll_make (2.0), execute_sim (3.2)                         */
/*  CALLING MODULES: none                                                    */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
void* main()
{

        struct driver* carwash;
        struct argument_list* argument;
        struct linked_list* sim_stats;
        int time1 = 0;                  /* time1 is the start time */
        int time2 = 50;                 /* time2 is the closure time */

        if ((argument=(struct argument_list*)malloc(sizeof(struct argument_list)))==NULL)
```

```
            return NULL;

        argument->time = &time1;
        argument->carwash = (carwash = make_driver(3, compare_time));

        line = li_make (FIFO);

        schedule_event (carwash, &time1, open_wash, argument);
        schedule_event (carwash, &time2, close_wash, carwash);
        sim_stats = execute_sim(carwash);
        /* print_stats (sim_stats) */
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: compare_time                                                       */
/*  MODULE NUMBER: 1.1                                                       */
/*  DESCRIPTION: Compare_time is used to tell sim_driver how to sort events  */
/*  ALGORITHM: return the resident time minus the time of the item to be     */
/*  inserted                                                                 */
/*  PASSED VARIABLES: *time1, *time2                                         */
/*  RETURNS: time2 - time1                                                   */
/*  GLOBAL VARIABLES USED: none                                              */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: none                                                     */
/*  CALLING MODULES: new_compare_time (3.0.1)                                */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
int compare_time (time1, time2)
int* time1;
int* time2;
{
        return (*time2 - *time1);
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: make_car_id                                                        */
/*  MODULE NUMBER: 1.2                                                       */
/*  DESCRIPTION: make_car_id is used to generate a car id #                  */
/*  ALGORITHM: every time the function is called increment the count         */
/*  PASSED VARIABLES: none                                                   */
/*  RETURNS: car_id                                                          */
/*  GLOBAL VARIABLES USED: none, but car_id is declared static in this func  */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: none                                                     */
/*  CALLING MODULES: open_wash (1.4), and car_arrives (1.5)                  */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
int make_car_id ()
{
```

```
        static int car_id = 0;
        return ++car_id;
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: close_wash                                                         */
/*  MODULE NUMBER: 1.3                                                       */
/*  DESCRIPTION: Signals carwash is closed, ends the simulation              */
/*  ALGORITHM: execute end_sim function                                      */
/*  PASSED VARIABLES: *car_wash                                              */
/*  RETURNS: none                                                            */
/*  GLOBAL VARIABLES USED: none                                              */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: end_sim (3.3)                                            */
/*  CALLING MODULES: execute_sim (3.2)                                       */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/
void close_wash (carwash)
struct driver* carwash;
{
        end_sim (carwash);
        printf ("SORRY THE CARWASH IS NOW CLOSED\n\n");
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: open_wash                                                          */
/*  MODULE NUMBER: 1.4                                                       */
/*  DESCRIPTION: signals that the carwash is open, schedules 1st car arrival */
/*  ALGORITHM: none                                                          */
/*  PASSED VARIABLES: *argument                                              */
/*  RETURNS: none, but does print a message to standard output               */
/*  GLOBAL VARIABLES USED: none                                              */
/*  GLOBAL VARIABLES CHANGED: none                                           */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED:      r_id (1.2), schedule_event (3.1)                    */
/*  CALLING MODULES: e    e_sim (3.2)                                        */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/

void open_wash (argument)
struct   gument_list* argument;
{
        int first_arrival;

        printf ("THE CARWASH IS NOW OPEN. TIME STAMP = %d\n", *argument->time);
        *argument->time = *argument->time + (rand() % 11) + 1;  /* creating next car_arrives time */
        argument->car_id = make_car_id();                       /* creating next car's id */
        schedule_event (argument->carwash, argument->time, car_arrives, argument);
}
/*****************************************************************************/
```

```
/*  DATE: 03/05/90                                                          */
/*  VERSION: 0.0                                                            */
/*  NAME: car_arrives                                                       */
/*  MODULE NUMBER: 1.5                                                      */
/*  DESCRIPTION: Will schedule a start_wash event if wash is empty, schedules */
/*               a car_arrives event                                        */
/*  ALGORITHM: if arrival time is after last wash exit time                 */
/*                     schedule a start_wash                                */
/*             else                                                         */
/*                     put the car in the line queue                        */
/*             schedule the next car_arrives event                          */
/*  PASSED VARIABLES: *argument                                             */
/*  RETURNS: none                                                           */
/*  GLOBAL VARIABLES USED: line, in_use_flag                                */
/*  GLOBAL VARIABLES CHANGED: may insert into line                          */
/*  FILES READ: none                                                        */
/*  FILES WRITTEN: none                                                     */
/*  HARDWARE INPUT: none                                                    */
/*  HARDWARE OUTPUT: none                                                   */
/*  MODULES CALLED: schedule_event (3.1), ll_insert (2.6), make_car_id (1.2) */
/*  CALLING MODULES: execute_sim (3.2)                                      */
/*  ORDER OF: This function is of order O(n) where n is the number of items */
/*            in line                                                       */
/*  AUTHOR: Capt Rob Rizza                                                  */
/*  HISTORY: none                                                           */
/*************************************************************************/
void* car_arrives (argument)
struct argument_list* argument;
{
        struct argument_list* new_argument;
        int new_car_id, time;
        int* time_ptr;

        printf ("CAR %d", argument->car_id);
        printf (" HAS ARRIVED AT THE CARWASH. TIME STAMP = %d\n", *argument->time);

        if (*argument->time > in_use_flag)   /* if wash is empty schedule immediate start_wash */
                schedule_event (argument->carwash, argument->time, start_wash, argument);
        else                                 /* else if wash busy put car in line */
                ll_insert (line, argument);

        if ((new_argument=(struct argument_list*)malloc(sizeof(struct argument_list)))==NULL)
                return NULL;
        if ((time_ptr=(int*)malloc(sizeof(time)))==NULL)    /* need to malloc for int time because */
                return NULL;                                /* don't want to over-write same space */
                                                            /* in memory                           */
        new_argument->time = time_ptr;
        *new_argument->time = *argument->time + (rand() % 11) + 1;  /* new car_arrives time for next car */
        new_argument->carwash = argument->carwash;
        new_argument->car_id = make_car_id();                       /* making new car_id for next car */

        schedule_event (new_argument->carwash, new_argument->time, car_arrives, new_argument);
}

/*************************************************************************/
/*  DATE: 03/05/90                                                          */
/*  VERSION: 0.0                                                            */
/*  NAME: start_wash                                                        */
/*  MODULE NUMBER: 1.6                                                      */
/*  DESCRIPTION: Signals start of wash.  Schedules end_wash event           */
/*  ALGORITHM: none                                                         */
/*  PASSED VARIABLES: *arguments                                            */
/*  RETURNS: none                                                           */
/*  GLOBAL VARIABLES USED: in_use_flag                                      */
/*  GLOBAL VARIABLES CHANGED: in_use_flag                                   */
```

```
/*  FILES READ: none                                                        */
/*  FILES WRITTEN: none                                                     */
/*  HARDWARE INPUT: none                                                    */
/*  HARDWARE OUTPUT: none                                                   */
/*  MODULES CALLED: schedule_event (3.1)                                    */
/*  CALLING MODULE: execute_sim (3.2)                                       */
/*  ORDER OF: This function is of order O(1)                                */
/*  AUTHOR: Capt Rob Rizza                                                  */
/*  HISTORY: none                                                           */
/**************************************************************************/
void start_wash (argument)
struct argument_list* argument;
{
        unsigned int j = 0;

        printf ("CAR %d", argument->car_id);
        printf (" HAS JUST ENTERED THE WASH. TIME STAMP= %d\n", *argument->time);
        while (j++ < 65000)             /* time to read screen loop */
                ;
        in_use_flag = (*argument->time = *argument->time + 5);
        schedule_event (argument->carwash, argument->time, end_wash, argument);
}

/**************************************************************************/
/*  DATE: 03/05/90                                                         */
/*  VERSION: 0.0                                                           */
/*  NAME: end_wash                                                         */
/*  MODULE NUMBER: 1.7                                                     */
/*  DESCRIPTION: Signals end of wash.  May schedule a rewash event or may  */
/*               schedule a start_wash event for the next car in line      */
/*  ALGORITHM: if a random # mod 5 = 3                                     */
/*                     schedule a rewash event                             */
/*             else if there is someone in line                            */
/*                     schedule a start_wash event                         */
/*  PASSED VARIABLES: *arguments                                           */
/*  RETURNS: none                                                          */
/*  GLOBAL VARIABLES USED: line                                            */
/*  GLOBAL VARIABLES CHANGED: may pop from line                            */
/*  FILES READ: none                                                       */
/*  FILES WRITTEN: none                                                    */
/*  HARDWARE INPUT: none                                                   */
/*  HARDWARE OUTPUT: none                                                  */
/*  MODULES CALLED: schedule_event (3.1), ll_isempty (2.4), ll_pop (2.1)   */
/*  CALLING MODULE: execute_sim (3.2)                                      */
/*  ORDER OF: This function is of order O(1)                               */
/*  AUTHOR: Capt Rob Rizza                                                 */
/*  HISTORY: none                                                          */
/**************************************************************************/
void end_wash (argument)
struct argument_list* argument;
{
        struct argument_list* temp_argument;
        unsigned int j = 0;
        printf ("CAR %d", argument->car_id);
        printf (" WASH IS FINISHED. TIME STAMP= %d\n", *argument->time);
        while (j++ < 65000)             /* time to read screen loop */
                ;
        if (rand() % 5 == 3)            /* random selection of rewashes */
                schedule_event (argument->carwash, argument->time, rewash, argument),

        else if (!ll_isempty(line))     /* if you don't get a rewash get next start_wash */
        {                               /* from the line if there's someone in it       */
                temp_argument = (ll_pop(line));
                argument->car_id = temp_argument->car_id;
                schedule_event (argument->carwash, argument->time, start_wash, argument),
```

```
        }
}

/*****************************************************************************/
/*  DATE: 03/05/90                                                           */
/*  VERSION: 0.0                                                             */
/*  NAME: rewash                                                             */
/*  MODULE NUMBER: 1.6                                                       */
/*  DESCRIPTION: Cars are in the process of being rewashed                   */
/*  ALGORITHM: none                                                          */
/*  PASSED VARIABLES: *arguments                                             */
/*  RETURNS: none                                                            */
/*  GLOBAL VARIABLES USED: in_use_flag                                       */
/*  GLOBAL VARIABLES CHANGED: in_use_flag                                    */
/*  FILES READ: none                                                         */
/*  FILES WRITTEN: none                                                      */
/*  HARDWARE INPUT: none                                                     */
/*  HARDWARE OUTPUT: none                                                    */
/*  MODULES CALLED: schedule_event (3.1)                                     */
/*  CALLING MODULES: execute_sim (3.2)                                       */
/*  ORDER OF: This function is of order O(1)                                 */
/*  AUTHOR: Capt Rob Rizza                                                   */
/*  HISTORY: none                                                            */
/*****************************************************************************/

void rewash (argument)
struct argument_list* argument;
{
        unsigned int j = 0;

        printf ("CAR %d", argument->car_id);
        printf (" HAS ENTERED FOR A REWASH. TIME STAMP= %d\n", *argument->time);
        while (j++ < 65000)             /* time to read screen loop */
                ;
        in_use_flag = (*argument->time = *argument->time + 5);
        schedule_event (argument->carwash, argument->time, end_wash, argument);
}
```

C.3.3 Script of Hogwash Execution

```
THE CARWASH IS NOW OPEN. TIME STAMP = 0
CAR 1 HAS ARRIVED AT THE CARWASH. TIME STAMP = 9
CAR 1 HAS JUST ENTERED THE WASH. TIME STAMP= 9
CAR 1 WASH IS FINISHED. TIME STAMP= 14
CAR 2 HAS ARRIVED AT THE CARWASH. TIME STAMP = 19
CAR 2 HAS JUST ENTERED THE WASH. TIME STAMP= 19
CAR 3 HAS ARRIVED AT THE CARWASH. TIME STAMP = 21
CAR 2 WASH IS FINISHED. TIME STAMP= 24
CAR 3 HAS JUST ENTERED THE WASH. TIME STAMP= 2[illegible]
CAR 4 HAS ARRIVED AT THE CARWASH. TIME STAMP = 2[illegible]
CAR 3 WASH IS FINISHED. TIME STAMP= 29
CAR 3 HAS ENTERED FOR A REWASH. TIME STAMP= 29
CAR 3 WASH IS FINISHED. TIME STAMP= 34
CAR 4 HAS JUST ENTERED THE WASH. TIME STAMP= 34
CAR 5 HAS ARRIVED AT THE CARWASH. TIME STAMP = 35
CAR 6 HAS ARRIVED AT THE CARWASH. TIME STAMP = 36
CAR 4 WASH IS FINISHED. TIME STAMP= 39
CAR 5 HAS JUST ENTERED THE WASH. TIME STAMP= 39
CAR 7 HAS ARRIVED AT THE CARWASH. TIME STAMP = 44
CAR 5 WASH IS FINISHED. TIME STAMP= 44
CAR 6 HAS JUST ENTERED THE WASH. TIME STAMP= 44
CAR 6 WASH IS FINISHED. TIME STAMP= 49
CAR 7 HAS JUST ENTERED THE WASH. TIME STAMP= 49
CAR 8 HAS ARRIVED AT THE CARWASH. TIME STAMP = 50
CAR 7 WASH IS FINISHED. TIME STAMP= 54
CAR 8 HAS JUST ENTERED THE WASH. TIME STAMP= 54
CAR 9 HAS ARRIVED AT THE CARWASH. TIME STAMP = 58
CAR 8 WASH IS FINISHED. TIME STAMP= 59
CAR 9 HAS JUST ENTERED THE WASH. TIME STAMP= 59
CAR 9 WASH IS FINISHED. TIME STAMP= 64
CAR 10 HAS ARRIVED AT THE CARWASH. TIME STAMP = 66
CAR 10 HAS JUST ENTERED THE WASH. TIME STAMP= 66
CAR 10 WASH IS FINISHED. TIME STAMP= 71
CAR 11 HAS ARRIVED AT THE CARWASH. TIME STAMP = 74
CAR 11 HAS JUST ENTERED THE WASH. TIME STAMP= 74
CAR 11 WASH IS FINISHED. TIME STAMP= 79
CAR 11 HAS ENTERED FOR A REWASH. TIME STAMP= 79
CAR 12 HAS ARRIVED AT THE CARWASH. TIME STAMP = 83
CAR 11 WASH IS FINISHED. TIME STAMP= 84
CAR 12 HAS JUST ENTERED THE WASH. TIME STAMP= 84
CAR 12 WASH IS FINISHED. TIME STAMP= 89
CAR 13 HAS ARRIVED AT THE CARWASH. TIME STAMP = 90
CAR 13 HAS JUST ENTERED THE WASH. TIME STAMP= 90
CAR 13 WASH IS FINISHED. TIME STAMP= 95
CAR 13 HAS ENTERED FOR A REWASH. TIME STAMP= 95
CAR 14 HAS ARRIVED AT THE CARWASH. TIME STAMP = 96
CAR 13 WASH IS FINISHED. TIME STAMP= 100
CAR 14 HAS JUST ENTERED THE WASH. TIME STAMP= 100
SORRY THE CARWASH IS NOW CLOSED
```

Appendix D. DISPLAY DRIVER INTERFACE REQUIREMENTS

This appendix is included for completeness of this document. It was taken directly from the thesis by DeRouchey (9).

The datafile is composed of records of several types. Each record type contains fields in a specific format. The number of fields in a record is different for each record type. In all cases the first field contains an integer which defines the record type.

Types

Icon Assignment Assigns an icon index to a viewable object.

```
30 O I
```

```
Example: 30 3 30
```

Table D.1. Record Type 30

Field	*Description*
30	Record Id
O	Object Index Number
I	Icon Index Number

Object numbers must begin with 1 and be sequential.

Object Location Contains position and orientation data for a viewable object. The position and velocity values have a maximum width of eleven characters. This width is inclusive of a minus sign and a decimal position. The angles are measured according to the right-hand rule, which is as follows: as you look down the positive rotation axis to the origin, positive rotation is counterclockwise.

```
31 0 T x y z Vx Vy Vz h p r Vh Vp Vr
```

```
Example: 31 2 2.5 1000 500 -20 1.2 2.4 -.3 30.0 60.0 -90.0 0.5 5.0 -1.0
```

Table D.2. Record Type 31

Field	*Description*
31	Record Id
O	Object Index Number
T	Time (seconds)
X	X - position (meters)
Y	Y - position (meters)
Z	Z - position (meters)
VX	velocity in x (meters/sec)
VY	velocity in y (meters/sec)
VZ	velocity in z (meters/sec)
H	Heading (degrees)
P	Pitch (degrees)
R	Roll (degrees)
VH	change in Heading (degrees/sec)
VP	change in Pitch (degrees/sec)
VR	change in Roll (degrees/sec)

Icon Identification Identifies an icon by index and geometry description filename.

```
32 I F
```

```
Example: 32 3 mig1
```

Icon number is determined freely by the user.

Table D.3. Record Type 32

Field	*Description*
32	Record Id
I	Icon Index Number
F	Icon Filename

Object Termination Identifies when an object is to be terminated.

```
33 O T
```

```
Example: 33 3 115.5
```

Table D.4. Record Type 33

Field	*Description*
33	Record Id
O	Object Index Number
T	Termination Time

Start Display Indicates all icons and the initial starting positions have been identified and sent to the graphics engine. The graphics engine can begin displaying the simulation.

```
50
```

```
Example: 50
```

Table D.5. Record Type 50

Field	*Description*
50	Record Id

Reset Display Indicates to the graphics display system that the simulation was restarted and will begin execution. The graphics display system will pause until a START DISPLAY is received.

```
52
```

```
Example: 52
```

Table D.6. Record Type 52

Field	*Description*
52	Record Id

End of Simulation Indicates the end of the simulation. This will be the last line within the datafile that is read.

```
86 T
```

```
Example: 86 245.0
```

Table D.7. Record Type 86

Field	*Description*
86	Record Id
T	Termination time

Ordering

All icon identifications (type 32) must occur before any other type of record in the datafile. Each viewable object must be associated with an icon (type 30) before a location record (type 31) for that object can occur in the datafile.

Appendix E. **RIZSIM Configuration Guide**

E.1 Introduction to the rizsim Configuration Guide

To run the rizsim simulation a number of supporting files need to be linked together. The files needed to be linked together are rizsim.c, ll.c, sim_driv.c, sim_func.c, and events.c. See Figure 3.6 for the file relationships. This configuration guide details the compiling order of the associated files to create the executable rizsim simulation code. The next section presents the UNIX makefile format used to compile and link the needed code. Other code which also must be present during the compile and link phase are ll.h, sim_driv.h, sim_func.h, and events.h.

E.2 Rizsim Makefile

```
CFLAGS = -g

OBJS = sim_driv.o events.o sim_func.o ll.o
LIB = -lm

rizsim: $(OBJS) rizsim.o
cc -o rizsim $(OBJS) $(LIB) rizsim.o

rizsim.o: rizsim.c
sim_driv.o: sim_driv.c
events.o: events.c
sim_func.o: sim_func.c
ll.o: ll.c
```

Appendix F. RIZSIM USERS GUIDE

It is not the intent of this appendix to describe the functions, events, or expected behavior of any particular simulation. It is assumed the user already knows how the simulation should perform given a starting scenario. The intent of this appendix is to describe how the input scenario file is created and named.

Currently, as the rizsim.c code specifies, the scenario input file must be named "datafile.c". If it becomes necessary to change the name of the input file it can be done by simply changing the read_datafile function call parameter in the rizsim.c code to correspond to the desired new data file name.

Figure F.1 shows the scenario input file format. **All field entries are mandatory with the exception of those fields which are shown as "can be repeated" fields.** These fields are directly tied the corresponding fields directly preceeding them, which gives the number of times the fields are to be repeated, if they are to be given at all. For example, if field 27 was a 5, then fields 28, 29, and 30 should be repeated five times to accomodate the five sensors. Conversely, if field 27 was a 0, no entries for fields 28, 29, or 30 would be given, and the next entry should be field 31.

Each field is seperated by a single space. A line of data encompasses all data needed for one object. A carraige return seperates lines of data, thus, carraige returns will be after either field 43 or 46.

A word of caution is appropriate at this point. Since there is a wide variety of legal enties for each field there has been no attempt to determine if any particular entry is correct. This translates to mean that although a created file may be correct format wise, it is up to the user to ensure that the data entered is correct. Incorrect input, if not caught before the simulation is displayed will undoubtably result in display anomalies. There is, however, some error checking being done on the input

Appendix F. RIZSIM USERS GUIDE

It is not the intent of this appendix to describe the functions, events, or expected behavior of any particular simulation. It is assumed the user already knows how the simulation should perform given a starting scenario. The intent of this appendix is to describe how the input scenario file is created and named.

Currently, as the rizsim.c code specifies, the scenario input file must be named "datafile.c". If it becomes necessary to change the name of the input file it can be done by simply changing the read_datafile function call parameter in the rizsim.c code to correspond to the desired new data file name.

Figure F.1 shows the scenario input file format. **All field entries are mandatory with the exception of those fields which are shown as "can be repeated" fields.** These fields are directly tied the corresponding fields directly preceeding them, which gives the number of times the fields are to be repeated, if they are to be given at all. For example, if field 27 was a 5, then fields 28, 29, and 30 should be repeated five times to accomodate the five sensors. Conversely, if field 27 was a 0, no entries for fields 28, 29, or 30 would be given, and the next entry should be field 31.

Each field is seperated by a single space. A line of data encompasses all data needed for one object. A carraige return seperates lines of data, thus, carraige returns will be after either field 43 or 46.

A word of caution is appropriate at this point. Since there is a wide variety of legal enties for each field there has been no attempt to determine if any particular entry is correct. This translates to mean that although a created file may be correct format wise, it is up to the user to ensure that the data entered is correct. Incorrect input, if not caught before the simulation is displayed will undoubtably result in display anomalies. There is, however, some error checking being done on the input

data file; specifically the number of fields are checked against those required (i.e. if a mandatory field is omitted or if an improper number of the optional fields are provided, an error message will appear on the screen).

Once the input scenario file is created and the executable rizsim code has been created, all that then needs to be done is to type the executable file name. The output is sent to a file called display.c in the directory where the executable code is run.

field 1	field 2	field 3	field 4	field 5	field 6	field 7
int	int	int	double	int	int	int
object type	object id	obj loyalty	curr time	fuel stat	condition	vulnerability

field 8	field 9	field 10	field 11	field 12	field 13	field 14
double	double	double	double	double	double	double
curr x coord	curr y coord	curr z coord	x velocity	y velocity	z velocity	yaw rate

field 15	field 16	field 17	field 18	field 19	field 20	field 21
double	double	int	int	int	int	int
pitch rate	roll rate	experience	threat know	min turn rad	max speed	ave fuel cons

field 22	field 23	field 24	field 25	field 26	field 27	field 28
int	int	double	double	double	int	int
max climb	# routepts	x coord	y coord	z coord	# sensors	sensor type

can be repeated # routept times

can be repeated

field 29	field 30	field 31	field 32	field 33	field 34	field 35
int	int	int	int	int	int	int
sensor range	sensor resolution	# armaments	arm type	arm range	arm yeild	arm accuracy

sensor times

can be repeated # armament times

field 36	field 37	field 38	field 39	field 40	field 41	field 42
int	int	int	int	double	double	double
arm speed	arm count	# targets	target type	targ x coord	targ y coord	targ z coord

can be repeated # target times

field 43	field 44	field 45	field 46
int	int	int	int
# defensive sys	def sys type	def sys range	def sys effect

can be repeated # defensive sys times

Figure F.1. Input File Format

Vita

Robert John Rizza was born February 24, 1957, in New York City. After graduating from Springfield Gardens High School in 1975, he enlisted in the United States Air Force. He seperated from the Air Force in 1979 and enrolled in the University of South Florida, Tampa. After receiving an Associate of Arts degree he transferred to the University of Central Florida, Orlando. In 1983 he received an ROTC commission and B.S. degree in environmental engineering technology, graduating summa cum laude. He served as a test manager at Wright Patterson AFB before attending the basic meteorology program at Texas A&M University in 1985. He served as the Wing Weather Officer to the 509th Bomb Wing, Pease AFB, prior to attending the Air Force Institute of Technology. He is married to Kathleen M. Rizza and has one son: Keith.

Permanent address: RD 2, PO Box 412
Cold Spring, New York
10516

Bibliography

1. "Military Make-believe," *Miltronics*, pages 16–24 (May 1989).
2. Battilega, John A and Judith K. Grange. *The Military Applications of Modelling*. Ohio: AFIT Press, 1981.
3. Bent, Nathaniel E. and Robert M. Kerchner. "The TAC Brawler Air Combat Simulation." In *Military Computing Conference*, pages 250–258, 1987.
4. Bezivin, Jean. "Some Experiments in Object-Oriented Simulation." In *Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) 87 Proceedings*, pages 394–405, 1987.
5. Bezivin, Jean. "Design and Implementation Issues in Object-Oriented Simulation," *Simuletter*, *19*:47 – 53 (1988).
6. Booch, Grady. *Software Components with ADA*. Menlo Park, CA: Benjamin/Cummings, 1987.
7. Booch, Grady. *Software Engineering with ADA 2nd Edition*. Menlo Park, CA: Benjamin/Cummings, 1988.
8. Chan, Stephen L. and Barbara J. Vogel. "Simulation of Multiple Aircraft Information, Communication, and Decision in Air Combat," *Mathmatics and Computer Modelling*, *11*:865–870 (1988).
9. DeRouchey, William. *A Remote Visual Interface Tool for Simulation Control and Display*. MS thesis, Air Force Institute of Technology, 1990. AFIT/GCS/ENG/90D-03.
10. Duncan, Ralph. "A Survey of Parallel Computer Architectures," *Computer*, pages 5–16 (February 1990).
11. Flynn, R. J. "Very High Speed Computing Systems," *IEEE Proceedings*, *54*:1901–1909 (1966).
12. Fujimoto, Richard M. "Parallel Discrete Event Simulation." In MacNair, E., et al., editors, *Proceedings of the 1989 Winter Simulation Conference*, pages 19 – 28, 1989.
13. Garrambone, Michael W., 1990. Major, USA, Operations Research Dept., Air Force Institute of Technology, Personal Interviews.
14. Goldberg, A. and D. Robson. *Smalltalk-80: The language and its Implementation*. Reading MA: Addison-Wesley, 1983.
15. Humphrey, Watts S. *Managing the Software Process*. New York: Addison-Wesley, 1989.

16. Jarmark, B. "Two Aggressive Aircraft in a Realistic Short-Range Combat as a Differential Game Study," *Computers and Mathmatics with Applications*, *18*:101–105 (1989).

17. Kaudel, Fred J. "A Literature Survey on Distributed Discete Event Simulation," *ACM SIGSIM Simuletter*, *18*(2):11–21 (June 1987).

18. Kernigan, Brian W. and Dennis M. Ritchie. *The C Programming Language*. New Jersey: Prentice Hall, 1988.

19. Linowes, Jonathan S. "It's an Attitude," *Byte*, *13*:219 - 224 (1988).

20. Neelamkavil, Francis. *Computer Simulation and Modelling*. John Wiley and Sons, 1987.

21. Nicols, David M. "Mapping a battlefield simulation onto message-passing parallel architectures," *Distributed Simulation*, pages 141–146 (1989).

22. Nygaard, K. and O.J. Dahl. "The Development of the Simula Language." In *Conference on History of Programming Languages*, 1978.

23. Pritsker, A. Alan B. and Claude D. Pegden. *Introduction to Simulation and SLAM*. New York: John Wiley and Sons, 1984.

24. Reynolds Jr., Paul F. and Phillip M. Dickens. *Spectrum: A Parallel Simulation Testbed*. Technical Report, Virginia Institute for Parallel Computation, 1988.

25. Roberts, Stephen D. and Joe Heim. "A Perspective on Object-Oriented Simulation." In *Proceedings of the 1989 Winter Simulation Conference*, pages 277 - 281, 1989.

26. Samuels, M. L. and J. R. Spiegal. "The Flexible ADA Simulation Tool (FAST) and its Extensions." In *Proceedings of the 1987 Winter Simulation Conference*, 1987.

27. Selvaraj, Sathyakumar, et al. "C Based Discrete Event Simulation Support System." In *Proceedings of the 1988 Winter Simulation Conference*, 1988.

28. Sommerville, Ian. *Software Engineering*. Addison - Wesley, 1989.

29. Thesen, Arne and Laurel Travis. "Introduction to Simulation." In *Proceedings of the 1989 Winter Simulation Conference*, pages 7 – 14, 1989.

30. Waite, Mitchell, et al. *The Waite Group's C Primer Plus*. Indiana: Howard w. Sams and Company, 1988.

31. Weiland, Frederick, et al. "An Imperical Study of Data Partitioning and Replication in Parallel Simulation." In *The Fifth Distributed Memory Computing Conference*, pages 915–921, April 1990.

32. White, Eric. "Object-Oriented Programming as a Programming Style," *The C Users Journal*, pages 43–58 (February 1990).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1990	Master's Thesis

4. TITLE AND SUBTITLE

AN OBJECT-ORIENTED MILITARY SIMULATION BASELINE FOR PARALLEL SIMULATION RESEARCH

5. FUNDING NUMBERS

6. AUTHOR(S)

Robert J. Rizza, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/90D-12

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This paper documents the design and implementation of a discrete event military simulation using a modula object-oriented design and the C programming language. The basic simulation is one of interacting objects. The objects move along a predetermined path until they encounter another object. Objects react to the encountere object according to the implemented algorithm. Object reaction options are fight, evade, or do nothing. In the code's current form it is generic enough to allow a user the flexibility of creating an infinite number of scenario bounded in size by the hardware's memory capacity. The modularity of design will allow for easy expansion o object complexity and detail, as well as easy removal or replacement of functions or events. The simulation code makes use of a generic linked list data structure and simulation driver. This adds yet another area to the code where expansion, removal, or replacement could be easily accomplished. The net result is a military scenari simulation program which is highly expandable and modifiable, yet compact enough to be easily understood.

14. SUBJECT TERMS

Military Simulation, Object-Oriented Simulation, Simulation

15. NUMBER OF PAGES

196

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL